uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería Informática

2018/2019

Trabajo de fin de grado

Buscador de documentos con imágenes de texto manuscrito mediante Machine Learning

---

Fernando Collado Egea

Tutor

Francisco Javier García Polo

Leganés, 16 de Junio, 2019

Índice

1. Introducción **1**
1.1. Descripción del problema . . . 1
1.2. Motivación . . . 2
1.3. Objetivos . . . 3
1.4. Estructura del documento . . . 5

2. Estado del arte **6**
2.1. Paradigma web . . . 6
2.1.1. Python . . . 8
2.1.2. Django . . . 12
2.1.3. React . . . 13
2.1.4. Base de datos . . . 16
2.2. Machine Learning . . . 17
2.2.1. Problemas en Machine Learning . . . 18
2.2.2. Clasificación de modelos de Machine Learning . . . 20
2.2.3. Neural Networks (NN) . . . 22
2.2.4. Reconocimiento de texto manuscrito . . . 25
2.3. Bibliotecas digitales . . . 30

3. Descripción del sistema **31**
3.1. Introducción . . . 31
3.2. Especificación de los casos de uso . . . 33
3.3. Descripción de los Casos de uso . . . 34
3.4. Requisitos . . . 39
3.4.1. Requisitos funcionales . . . 39
3.4.2. Requisitos no funcionales . . . 44
3.4.3. Matriz de trazabilidad de casos de uso . . . 46

4. Diseño del sistema **47**
4.1. Arquitectura del sistema . . . 47
4.2. Back - Django-Rest-Framework . . . 48
4.2.1. ”Flujo” de Django . . . 49
4.2.2. Modelos . . . 51
4.2.3. Serializers . . . 54
4.2.4. Views . . . 56
4.2.5. Endpoints . . . 61

4.2.6. Seguridad . . . 62
4.3. Front - React . . . 65
4.3.1. Header . . . 66
4.3.2. Landing . . . 67
4.3.3. Log in . . . 68
4.3.4. Search . . . 68
4.3.5. Profile . . . 71
4.3.6. Components . . . 78
4.4. Machine Learning . . . 82

5. Experimentación 86
5.1. Entrevistas de usabilidad . . . 86
5.2. Machine Learning . . . 91

6. Organización del proyecto 97
6.1. Planificación . . . 97
6.1.1. Planificación de tiempo . . . 97
6.1.2. Control de versiones . . . 98
6.1.3. Presupuesto . . . 98
6.2. Marco regulador . . . 100
6.2.1. Protección de datos . . . 100
6.2.2. Derechos de autor . . . 102
6.3. Contexto socioeconómico . . . 102

7. Conclusiones y trabajo futuro 104
7.1. Conclusiones . . . 104
7.2. Trabajo futuro . . . 104

Anexos 108

Bibliografía 127

Índice de tablas

1. Acrónimos utilizados en este documento . . .
2. Plantilla de casos de uso . . . 35
3. Caso de usuario `CU-01` . . . 35
4. Caso de usuario `CU-02` . . . 35
5. Caso de usuario `CU-03` . . . 36

6. Caso de usuario CU-04 . . . 36
7. Caso de usuario CU-05 . . . 36
8. Caso de usuario CU-06 . . . 36
9. Caso de usuario CU-07 . . . 37
10. Caso de usuario CU-08 . . . 37
11. Caso de usuario CU-09 . . . 37
12. Caso de usuario CU-10 . . . 37
13. Caso de usuario CU-11 . . . 38
14. Plantilla de requisitos . . . 39
15. Requisito funcional RF-01 . . . 39
16. Requisito funcional RF-02 . . . 39
17. Requisito funcional RF-03 . . . 40
18. Requisito funcional RF-04 . . . 40
19. Requisito funcional RF-05 . . . 40
20. Requisito funcional RF-06 . . . 40
21. Requisito funcional RF-07 . . . 41
22. Requisito funcional RF-08 . . . 41
23. Requisito funcional RF-09 . . . 41
24. Requisito funcional RF-10 . . . 41
25. Requisito funcional RF-11 . . . 42
26. Requisito funcional RF-12 . . . 42
27. Requisito funcional RF-13 . . . 42
28. Requisito funcional RF-14 . . . 42
29. Requisito funcional RF-15 . . . 43
30. Requisito funcional RF-16 . . . 43
31. Requisito funcional RF-17 . . . 43
32. Requisito funcional RF-18 . . . 43
33. Requisito funcional RF-19 . . . 44
34. Requisito funcional RF-20 . . . 44
35. Requisito no funcional RNF-01 . . . 44
36. Requisito no funcional RNF-02 . . . 44
37. Requisito no funcional RNF-03 . . . 45
38. Requisito no funcional RNF-04 . . . 45
39. Requisito no funcional RNF-05 . . . 45
40. Requisito no funcional RNF-06 . . . 45
41. Matriz de trazabilidad de casos de uso . . . 46
42. Test de experiencia de usuario . . . 88
43. Resultados test cuantitativo a usuarios . . . 90

44. Resultados de Jaccard Similarity . . . 95
45. Resultados de Cosine Similarity . . . 95
46. Resultados de Damerau-Levenshtein . . . 96
47. Tiempo por posición planificado . . . 99
48. Coste de personal . . . 99
49. Costes de equipamiento . . . 99
50. Coste total del proyecto . . . 100
51. Jaccard Similarity results . . . 121
52. Cosine Similarity results . . . 121
53. Damerau-Levenshtein results . . . 121
54. Total project cost . . . 123

Índice de figuras

1. Interacción del MVC . . . 6
2. Diferencias en búsqueda de google trend entre MVC, React y Angular desde el año 2004 hasta hoy . . . 7
3. Diferencias entre JSX y non-JSX . . . 14
4. Sugerencia de algoritmos a utilizar en función de la tarea . . . 19
5. Perceptrón . . . 22
6. Gradient descent . . . 23
7. Multilayer Perceptron . . . 24
8. Buscador de manuscritos de la Biblioteca Nacional . . . 31
9. Diagrama de Casos de Uso para usuario y administrador . . . 33
10. Diagrama de Casos de Uso para administrador . . . 34
11. Arquitectura del sistema . . . 47
12. Sistema de ficheros en Django . . . 48
13. Modelo de base de datos . . . 51
14. Diagrama de vistas . . . 66
15. Menú de usuario tras hacer login . . . 67
16. Landing de la interfaz . . . 67
17. Login de la interfaz . . . 68
18. Vista Search de la interfaz . . . 69
19. Imagen resaltada que contiene el texto buscado . . . 70
20. Lightbox abierto con una de las imágenes que contienen el texto encontrado . . . 70
21. Lightbox abierto con una de las imágenes que **no** contienen el texto encontrado . . . 70

22. Profile en la interfaz . . . 71
23. Modal editar registro . . . 72
24. Modal añadir imágenes a un registro . . . 73
25. Modal editar imágenes de un registro . . . 74
26. Formulario para añadir un registro . . . 75
27. Formulario relleno para añadir un registro . . . 75
28. Mensaje de error por fecha incorrecta . . . 76
29. Listado de usuarios en Profile . . . 77
30. Modal para editar usuario . . . 77
31. Añadir un usuario . . . 78
32. Diagrama de componentes del portal . . . 79
33. Redirecciones desde Header a vistas . . . 81
34. Pasos necesarios para usar Google API Vision . . . 83
35. Pasos necesarios para usar una Service Account . . . 83
36. Imagen para test ML 1 . . . 93
37. Imagen para test ML 2 . . . 94
38. Imagen para test ML 3 . . . 94
39. Imagen para test ML 4 . . . 94
40. Imagen para test ML 5 . . . 94
41. Planificación del proyecto . . . 98
42. System Architecture . . . 115
43. Database model . . . 116
44. View Diagram . . . 118

Índice de ejemplos de código

1. Dumb component para un botón . . . 14
2. Smart component . . . 14
3. Modelo de Person en Django . . . 49
4. Tabla de Person en SQL . . . 49
5. Modelo Entry en Django . . . 52
6. Tabla Entry en SQL . . . 52
7. Modelo Image en Django . . . 53
8. Tabla Image en SQL . . . 53
9. ImageSerializer . . . 54
10. EntrySerializer . . . 55
11. UserSerializer . . . 55
12. UserDetailedSerializer . . . 55

13. Query del texto recibido 57
14. Queries de filtros adicionales 58
15. Ordenación en queries 58
16. UserView 60
17. CSRF attack 63
18. Exportar clave de acceso a Google Vision API 84
19. Uso de Vision API para detectar texto 84
20. Vision API en *settings.py* 85
21. Django Person model 115
22. SQL Person Table 116
23. Vision API in *settings.py* 119

Resumen

En un mundo cada vez más digitalizado, son cada vez más los organismos que hacen públicos a través de Internet sus documentos manuscritos. Libros, apuntes, diarios, registros de natalidad. Existe una gran variedad de estos documentos accesible al público en general. Muchas de las herramientas que gestionan estos documentos permiten búsquedas por título, autor, fecha... Sin embargo, existen pocas herramientas de búsqueda de texto dentro de estos documentos, por la complejidad y tediosidad de transcribir dichos textos, que normalmente se encuentran en forma de imágenes. Una herramienta que provea de esta capacidad tiene potencial de ser útil en diferentes ámbitos. Evitaría tener que hojear una a una las páginas de estos documentos hasta encontrar el contenido deseado. Por esta razón, en este proyecto se ha desarrollado una herramienta que permite tanto la gestión de estos documentos, como la búsqueda por texto en los mismos.

En particular, este trabajo detalla el estudio realizado sobre tecnologías existentes que sirven para acometer el problema descrito. Además, desde un punto de vista de ingeniería de software en este trabajo se detalla el diseño del sistema, implentación de los diferentes aspectos necesarios, y validación de la herramienta. También se han estudiado diferentes técnicas de Machine Learning que permiten traducir las imágenes que componen estos documentos a texto, y se han incorporado dichas técnicas a la herramienta propuesta. Por último, desde un punto de vista más académico, se provee de un estudio del transfondo individual de las tecnologías usadas.

Keywords: *Web, Django, React, Python, JavaScript, User Interface, User Experience, Machine Learning*

Abstract

In an ever more digitalized world, it is more and more common organizations that provide access to their handwritten documents. Books, notes, diaries, birth certificates. There exists a variety of such documents accessible to the general public. Many of the tools that manage these documents allow search by title, author, year... However, few of these tools allow search by text within the document, due to the complexity and tediousness of transcribing these texts, that usually are in image format. A tool that provides this capability has potential to be useful for academic and legislative purposes. It would avoid the need to go page by page in order to find the desired content. That is why a tool has been developed, one that allows both handling documents and searching by text through them.

Particularly, this document details the study made on existing technologies that serve this purpose. From a software engineering point of view, it is detailed in this document the design of the system, its development, and validation. Furthermore, different Machine Learning techniques that allow translating image to text have been researched, and said techniques have been incorporated to the tool. Lastly, from a more academic point of view, a research on technologies currently used for some of these purposes individually has been made.

Keywords: *Web, Django, React, Python, JavaScript, User Interface, User Experience, Machine Learning*

Acrónimos

Acrónimo	Significado
API	Application Programming Interface
CSS	Cascading Style Sheets
CSR	Client Server Rendering
CRUD	Create React Update Delete
DRF	Django-Rest-Framework
HTML	HyperText Markup Language
JS	JavaScript
LOPD	Ley Orgánica de Protección de Datos
LOPDGDD	LOPD y Garantía de Derechos Digitales
NN	Neural Net
ML	Machine Learning
MVP	Minimum Viable Product
MVC	Model View Controller
OCR	Optical Character Recognition
ORM	Object Relational Mapping
SPA	Single Page Application
SSR	Server Side Rendering
RGPD	Reglamento General de Protección de Datos
UI	User Interface
UX	User Experience

Table 1: Acrónimos utilizados en este documento

Agradecimientos

Gracias a mi familia, por su paciencia y apoyo a lo largo de mi educación.

A mi tutor, Francisco Javier García Polo, por la ayuda a lo largo del desarrollo de este trabajo.

Por último, a mis amigos Krzys, y Silvia, de gran ayuda a lo largo de este último paso.

1. Introducción

1.1. Descripción del problema

En un mundo cada vez más digitalizado, todavía se puede encontrar texto manuscrito en diferentes ámbitos. Libros que no se han transcrito nunca, registros eclesiásticos de natalidad, defunción y casamiento, apuntes, tanto de estudiantes actuales, como de personalidades a lo largo de la historia (e.g., Leonardo Da Vinci, Miguel de Cervantes). Sin embargo, son cada vez más los esfuerzo de bibliotecas y organismos de digitalizar este tipo de documentos. De esta manera, se hacen accesibles al público en general a través de Internet. Ejemplos de estos esfuerzos son los de la Biblioteca Nacional de España [BNE, 2019], Centro de Física "Miguel Antonio Catalán"[CSIC, 2017], el Museo del Prado [MP, 2019], la Biblioteca de Catalunya [BC, 2019], o los de diferentes organismos y asociaciones cuyo cometido es compartir apuntes digitalizados de diferentes carreras universitarias [BiblioFCC, 2019].

No obstante, si bien buscar estos documentos manuscritos por título, autor, fecha, no resulta complicado en los motores de búsqueda que ofrecen esas bibliotecas y organismos, el problema surge cuando se quieren realizar búsquedas más avanzadas, por ejemplo, la búsqueda de una palabra o frase dentro de estos documentos. En general, no queremos perder el tiempo hojeando las páginas una a una de estos (generalmente enormes) volúmenes hasta encontrar el contenido en concreto que buscamos: sería mucho mejor ofrecer al usuario la posibilidad de realizar búsquedas de palabras o frases dentro de las páginas de estos documentos de forma que pueda encontrar rápidamente el contenido que desea. Pero, ¿cómo hacerlo? En la mayoría de los casos estos documentos son una sucesión de imágenes, cada una con una página del manuscrito, lo que dificulta hacer la búsqueda sobre las palabras que contiene.

Actualmente, existen multitud de programas y técnicas para extraer texto manuscrito o incluso texto “a máquina” desde una imagen. La más usada es la Optical Character Recognition (OCR) que puede basarse en plantillas, reglas lógicas, y actualmente en técnicas de Machine Learning [Verma and Ali, 2012, Øivind Due Trier et al., 1996, Impedovo et al., 1991]. Sin embargo, en este trabajo se investiga el uso de aquellas técnicas basadas en Machine Learning para extraer el texto de estas imágenes.

Aún así, en la mayoría de los casos hay que transformar estas imágenes a texto de una en una, lo cual puede ser tedioso, o simplemente hacer desistir, a la persona que tenga que realizar dicha tarea. Adicionalmente, es difícil registrar este tipo de archivos, por ejemplo, en una base de datos de una biblioteca, ya que de nuevo, las tareas hay que hacerlas de una en una sin el apoyo de una herramienta que facilite el proceso.

En este trabajo se propone una solución para este problema, que puede abarcar los distintos ámbitos mencionados previamente. En particular, en este proyecto se pretende elaborar una herramienta que permita almacenar este tipo de documentos en una plataforma centralizada, extraer el texto tanto *manuscrito* como a máquina de estas imágenes mediante técnicas de Machine Learning y, lo más importante, permitir realizar búsquedas sobre ese texto extraído presentando los resultados de forma adecuada al usuario.

El desarrollo de esta plataforma ha permitido asentar una buena base sobre como llevar proyectos de esta índole desde un punto de vista individualizado, tras aplicar conocimientos adquiridos en proyectos durante mi vida profesional.

1.2. Motivación

Por lo tanto, la principal motivación del trabajo es facilitar hacer búsquedas dentro del contenido dentro de estos documentos manuscritos digitalizados, puesto que normalmente sólo se puede buscar por título, autor, fecha, pero no por contenido [BNE, 2019]. Poder hacer búsquedas de palabras/frases dentro del contenido facilitaría enormemente navegar entre las hojas de estos manuscritos. Además, se hace necesario construir un portal web amigable que permita envolver esta funcionalidad, y que permita también a los diferentes organismos e instituciones gestionar (crear, modificar, eliminar) los documentos digitalizados de forma sencilla.

Por otro lado, la inteligencia artificial me ha interesado desde que cursé mi primera asignatura de contenido similar, e incrementó mucho más desde que cursé estando de intercambio asignaturas ”avanzadas” en el mismo terreno. El desarrollo web empecé a cogerle el gusto aprendiendo por mi cuenta, y se afianzó en mis experiencias profesionales.

Por estos motivos, encontrar un trabajo que pudiera combinar ambos era para

mi la perfecta elección. Además, el trabajo de fin de grado en cuestión es una propuesta abierta, cuya definición inicial no comprende ninguna condicion técnica impuesta. La temática en sí resulta bastante atractiva, y el hecho de tener poder de decisión sobre cómo desarrollar el proyecto es muy atractivo.

Adicionalmente, pienso que el desarrollo de este proyecto (y un futuro posible lanzamiento a diferentes públicos) puede servir de gran ayuda a mucha gente, en su día a día del trabajo, o en encontrar información de forma más fácil y rápida que revisar uno a uno, y en persona, diferentes documentos.

1.3. Objetivos

El objetivo principal de este trabajo consiste en desarrollar una aplicación que permita gestionar estos documentos manuscritos, que permita crearlos, modificarlos, eliminarlos, y realizar búsquedas tanto por los campos habituales de título, autor, fecha, como dentro del contenido de los mismos. Para esto, es necesario dividir el trabajo en diferentes tareas, ya que no solo hay que realizar un portal de búsqueda, también se ha de implementar un Content Management System (CMS) para los registros sobre los que se hace la búsqueda.

1. **Toma de requisitos**
   Dado que la descripción inicial del trabajo permite gran libertad de desarrollo, es necesario empezar con una toma inicial de requisitos, para tener una idea exacta de todo lo que será necesario implementar en el sistema. Una vez un registro ha sido creado, ¿se puede eliminar? ¿Se puede, además, editar? ¿Quién lo puede editar? Son preguntas de gran importancia.

2. **Análisis de tecnologías web existentes**
   Las tecnologías web es uno de los dominios más cambiantes actuales en el mundo del desarrollo, a pesar de ser relativamente reciente. Cada 5-10 años surge un nuevo paradigma, una nueva forma de hacer las cosas. Hace 10 años, la práctica más extendida era hacer Server-Side-Rendering (SSR), renderizar el contenido en el servidor, mientras que ahora parece que cada vez más está orientado a Client-Side-Rendering (CSR), renderizar el contenido en el cliente.
   En resumen, CSR provee una gran usabilidad para el usuario, al no tener que refrescar la página con cada pequeña interacción (Trello es un ejemplo de CSR, si cada vez que un usuario en Trello moviese una tarjeta o estado se recargase la página entera no muchas personas la usarían), mientras que

SSR es más útil para páginas en las que hay pocas interacciones de usuario, por ejemplo, una página de blogging.
Es importante hacer un pequeño análisis de qué es lo que va a funcionar mejor para el proyecto, en funcioń de las tecnologías actuales. Este cambio de SSR a CSR implica un desacoplamiento de funcionalidades y tecnologías que también es necesario aprender.

3. **Análisis de tecnologías Machine Learning existentes**
   El proyecto en cuestión, en cuanto a Machine Learning se refiere, trata de reconocer texto (manuscrito) en imágenes. Es necesario realizar un análisis de qué tecnologías existentes nos permiten realizar dicha tarea, ya que existe una diferencia entre clasificar texto, detectar texto, y reconocer (texto) [Fletcher, 2019].

4. **Diseño del sistema**
   Una vez hecho todo el análisis previo de tecnologías en diferentes ámbitos, incluyendo la elección de tecnologías en concreto, y hecha la recogida de los requisitos, es necesario hacer el diseño del sistema. En este proyecto, se realizará un desacoplamiento de la parte del sistema que renderiza, y la parte del sistema que provee los datos. Es decir, habrá un *front* hecho en React, que se encargará de servir como interfaz para los usuarios para realizar distintas operaciones (búsqueda, creación, edición...) por medio de peticiones al servidor web, habrá una Application Programming Interface (API) que haga las veces de servidor web, que se encargará de recibir esas peticiones, y este a su vez, se encargará de proveer los datos desde la Base de Datos. Todo esto implica un diseño de endpoints en la API a los que el front tiene que llamar, una forma específica de estructurar las peticiones desde el front a la API... Por lo que es necesario detallar el diseño del sistema para tener una visión estructurada a la hora de implementar.

5. **Construcción del sistema** Este sistema permitirá gestionar fácilmente los documentos registrados en la aplicación. Permitirá, además, crear nuevos registros con documentos digitalizados, modificarlos y eliminarlos. Permitirá hacer búsquedas por los campos tradicionales de autor, fecha y título, y además permitirá buscar palabras/frases dentro de los documentos. Una vez realizada la búsqueda, se mostrará al usuario por cada registro las páginas que contienen la palabra/frase buscada de forma destacada, para que pueda identificar claramente dentro del manuscrito las páginas que contienen esa palabra/frase.

6. **Experimentación**
   Se deben experimentar dos cosas diferentes en este proyecto:

   - Machine Learning, diferentes técnicas tienen diferentes resultados, se elegirá la mejor en función de la mejor precisión.
   - Portal web, algo importante siempre que se desarrolla una aplicación web es testear la usabilidad del mismo. La experiencia de usuario (UX) siempre es un factor determinante para conocer si lo desarrollado es *usable* por los usuarios, o si habrá funcionalidades que no se usarán nunca debido a que no se conocen, o son difíciles de ”alcanzar”.

7. **Planificación**
   Para llevar a cabo este proyecto, es necesaria una planificación previa, en la que se estudiará el presupuesto del proyecto, la planificación de tiempo estimada, el impacto socioeconómico que este sistema puede tener, y por último el marco legal bajo el que se sostiene.

1.4. Estructura del documento

El documento está dividido en las siguientes secciones:

1. **Estado del arte**. Descripción del marco teórico en el cual se encuadra este proyecto.

2. **Descripción del sistema**. Se recogen los casos de usuario y requisitos del sistema.

3. **Implementación**. Se desarrolla la implementación llevada a cabo siguiendo la descripción del sistema.

4. **Experimentación**. Se recoge en esta sección las entrevistas a usuario que sirven para comprobar la usabilidad, y la comparativa entre diferentes técnicas de Machine Learning.

5. **Organización del proyecto**. Se detallan la planificación de tiempo para el proyecto, el presupuesto, y se describe el marco regulador y el ámbito socioeconómico.

6. **Conclusiones y trabajos futuros**. Conclusiones personales y propuestas de mejora.

7. **Anexos** Documentación del proyecto.

2. Estado del arte

Antes de entrar en materia con el diseño y la implementación, es importante exponer el estado del arte para entender el contexto sobre el cual este trabajo ha sido realizado. Empezando por los paradigmas web del pasado y los usados en el presente, al igual que un análisis de las herramientas usadas para este trabajo (Sección 2.1). Más adelante se describirá el estado del arte en reconocimiento de texto (tanto manuscrito como no) para entender mejor las diferentes técnicas existentes, sobre todo aquellas referidas a la utilización de Machine Learning, y seleccionar aquellas que mejor se adapten a los objetivos concretos de este trabajo (Sección 2.2). Para terminar, se describirán algunas *bibliotecas digitales* existentes y sus carencias (Sección 2.3).

2.1. Paradigma web

Las aplicaciones web han seguido típicamente la arquitectura Modelo Cliente Servidor (MVC), en la que un cliente realiza una serie de peticiones, y un (o más) servidores aplica lógica en función de dichas peticiones, y devuelve una respuesta.

Dentro de esta arquitectura, existen diferentes patrones de diseño, el más común hasta ahora ha sido Modelo Vista Controlador (MVC). El *modelo* se refiere a la funcionalidad básica, y el manejo de dato; la *vista* comprende mostrar la información al usuario; finalmente el *controlador* maneja el input del usuario, que a su vez manipula el modelo. En la figura 1 [1] se puede ver cómo funcionan estos tres elementos.

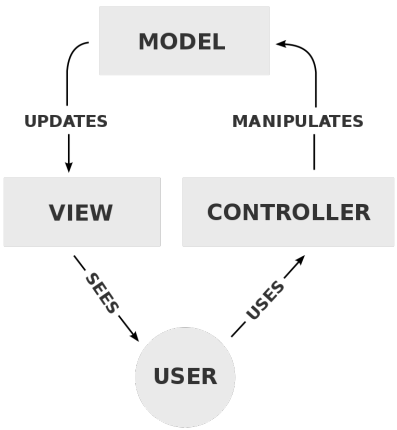

Figura 1: Interacción del MVC

[1] Imagen extraída de *https* : *//commons.wikimedia.org/wiki/File* : *MVC* − *Process.png*

Recientemente, un nuevo paradigma se está utilizando cada vez más y más, como se puede ver en la figura 2 (búsquedas en google desde el año 2004 hasta hoy). Este nuevo paradigma supone un desacoplamiento de la parte que maneja la funcionalidad y los datos, de la parte que visualiza dichos datos y la que controla el input del usuario [Marcio Galli and Oeschger, 2003]. A efectos prácticos, esto supone que el patrón de diseño MVC se divide en dos: una *Application Programming Interface (API)*, que puede seguir a su vez diferentes paradigmas, donde se manejan los datos, permisos, accesos...y una *Single Page Application (SPA)*, que se encarga de mostrar datos, y de manejar la forma de añadir, modificar y/o borrar dichos datos.

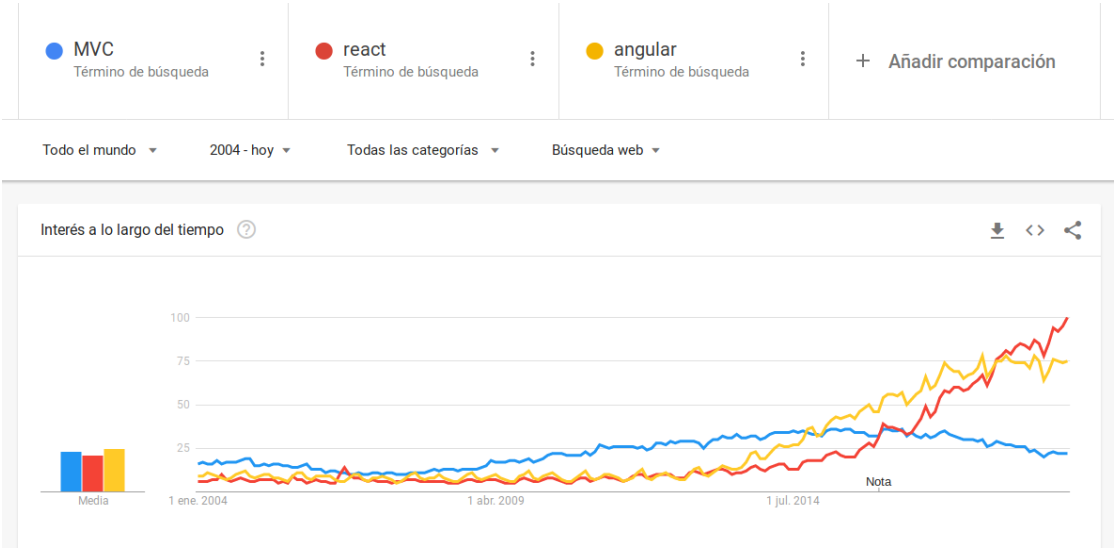

Figura 2: Diferencias en búsqueda de google trend entre MVC, React y Angular desde el año 2004 hasta hoy

Los navegadores funcionan con 3 tecnologías principales:

- *HTML* (Hypertext Markup Language)
  Establece la estructura de la página, mediante diferentes etiquetas se puede crear diferentes elementos, como pueden ser, * * para crear un contenedor, *<a>* para un hipervínculo, *<img>* para una imagen, entre otros. Cuando una página ha sido descargada, el navegador crea el Document Object Model (DOM), el cual es un árbol de objetos, cuyo contenido se basa en la jerarquía desarrollada en el documento HTML que esté cargando.

- *CSS* (Cascading Style Sheets)
  Describe el estilo de los diferentes elementos HTML, incluyendo por ejemplo, color, tamaño, animaciones...

- *JavaScript*
  Lenguaje de programación que permite manejar los elementos del DOM, modificarlos, eliminar o crearlos... Además de realizar peticiones HTTP.

Siguiendo ambos patrones de diseño anteriomente, en el patrón MVC todo este contenido, típicamente en ficheros .html, .css, y .js, carga directamente al cliente desde el servidor que contiene el modelo. Se podría decir que es un sistema "monolítico". Ese mismo servidor tendrá a su vez otro lenguaje de programación que se encargará de la funcionalidad y trata de datos. Algunos ejemplos de lenguajes de servidor web son lenguajes de scripting (PHP, Ruby, Python), o compilados (Java, Go).

En cambio, en el modelo desacoplado, el servidor web tiene contenido parecido al modelo MVC, pero nunca va a servir contenido HTML, CSS o JavaScript, simplemente hace de interfaz entre un cliente, y modelo. El equivalente a contenidos de Vista y Controlador, en este sistema desacoplado, vendrían dados por otro servidor distinto, cuya finalidad es proveer esos contenidos. El cliente nunca establecerá una petición a este servidor.

Adicionalmente, SPA se basa en mandar los datos una sola vez, todo el contenido web está hecho de forma que, aunque navegue por dentro de la página, no hay que hacer petición a un servidor para recibir nuevo contenido, está todo dado en una sola llamada. Este sistema desacoplado presenta varias ventajas, en las cuales se ahondan más profundamente en la descripción del sistema, sección 3.

A continuación, se hace un breve repaso de las tecnologías utilizadas para el portal web, con un enfoque desacoplado usando Python y React, con PostgreSQL como base de datos.

2.1.1. Python

Python es un lenguaje de programación creado por Guido Van Rossum, cuya primera versión data en 1991, pero no fue hasta 1994 cuando la versión 1.0 fue lanzada [Jaworski and Ziadé, 2016]. La versión actual de Python es la 3, teniendo la versión 2 fecha de deprecación el 1 de Enero de 2020.
Una de las características principales de Python es su sencillez de uso, y "parecido" con pseudocódigo, lo que permite además ser leído con gran facilidad. Esto permite una velocidad de desarrollo que otros lenguajes carecen.

Las características de Python son:

1. Es un lenguaje multi-paradigma, ya que soporta programación orientada a objectos, y estructura. Tiene además soporte para programación funcional, y programación orientada a aspectos

2. Tipado dinámico. Las variables no necesitan especificar tipo (int, String, boolean...) el intérprete de Python infiere el tipo durante la ejecución

3. Es un lenguaje interpretado, por lo que no hace falta compilar para poder ejecutar

4. Existen variaciones de Python en función de la necesidad, por ejemplo, *PyPy* es una versión de Python compilada con un compilador JIT (Just In Time), o *Cython*, que traduce Python a C

5. Una parte importante de la filosofía del lenguaje viene dada por la intención de hacer el lenguaje divertido de usar. Esto se refleja en el nombre, que viene dado por el grupo cómico que realizaba sketchs Monty Python. Otro ejemplo, el Zen de Python , PEP 20 (Python Enhancement Proposals), que habla, mediante un poema, de buenas prácticas a la hora de programar en Python

Como todo, este lenguaje tiene sus beneficios:

- Rápido desarrollo por la facilidad de uso, y de aprendizaje. Su fácil sintaxis muy parecida a pseudo código permite aprender de forma intuitiva el lenguaje.

- Gran cantidad de librerías, al ser uno de los lenguajes más comunes hoy en día [StackOverflow, 2019], con un 41 % de desarrolladores en uno de los forums más comunes usando Python.

- Buena cooperación con otros lenguajes, Python permite [WikiPython, 2019] la integración con hasta otros 16 lenguajes de programación.

- Multiparadigma que permite flexibilidad a la hora de usar el lenguaje. Se refiere al hecho de poder realizar, con el mismo lenguaje de programación, tareas tan diversas como implementar una aplicación web, entrenar un modelo predictivo de inteligencia artificial, crear un script para automatizar una tarea, manejar y modificar imágenes...

- Amplia documentación, muy bien diferenciada entre versiones y subversiones. Python es un lenguaje que está actualizandose constantemente, en ocasiones, de una versión a otra la implementación de una función de la librería

estandar cambia, y es necesario poder reflejar esos cambios. En la página de documentación de Python, para cambiar la versión de la documentación es tan sencillo como hacer literalmente dos clicks.

y desventajas:

- Lentitud: por lo general los lenguajes interpretados tienden a ser más lentos, aunque Python solventa este problema con sus variaciones PyPy y Cython
- Multiversión: al ser un lenguaje tan extendido, durante los primeros años de la versión 3, el número de librerías era escaso, al estar todo el mundo usando todavía la versión 2. Dentro de poco la versión 4 va a ser lanzada, y probablemente ocurra algo parecido.
- Errores durante ejecución, en lugar de errores de compilación. Uno de los inherentes rasgos de un lenguaje interpretado es que los errores (que no sean por sintaxis) van a aparecer durante la ejecución del programa, lo cual puede dificultar corregirlos.
- Débil en el aspecto móvil, si bien es posible desarrollar aplicaciones móviles en Python, la gran mayoría de los desarrolladores usan Android-Java, Kotlin, C...

Las ventajas superan a las desventajas para la mayoría de los escenarios en un entorno de desarrollo software, y es por esto que Python se usa en una gran variedad de campos, como pueden ser Desarrollo Web, Computación científica y numérica, Inteligencia artificial y Machine Learning, Computer Vision, desarrollo de aplicaciones de escritorio mediante sus módulos GUI, en educación...

El hecho de que Python sirve tanto para Machine Learning como para Desarrollo Web ha sido el *principal motivo para la elección de Python* para la elaboración de este trabajo, ya que permite una integración entre ambos servicios sin grandes problemas.

Una vez elegido el lenguaje de programación, lo siguiente a tener en cuenta es el framework web a usar. Un framework web es un framework que está diseñado para ayudar al desarrollo de aplicaciones web, incluyendo servicios web, y APIs. Dentro de esa ayuda, se incluyen librerías, y facilidad de uso a la hora de conectar el servicio que esté siendo desarrollado con acceso a base de datos, frameworks de plantillas HTML, y manejo de sesión de usuarios.

Dentro de Python, los frameworks web se dividen entre framework full-stack, y framework no full-stack. Full stack se refiere al dominio tanto back como front en una aplicación web. En cambio, un framework no full-stack, solo provee la parte back, pero no la parte front, por lo que es necesario un servicio adicional.

Los frameworks full stack más populares en python son:

- Django, un framework "con pilas incluidas", es decir, por defecto está configurado para incluir todo lo normal en un framework web, manejo de sesión, sistema de plantillas, *hooks* con varias bases de datos distintas, sin tener que instalar nada adicional. Versión estable 2.2, con última fecha de actualización el 1 de Abril de 2019, y una gran comunidad de desarrolladores
- TurboGears, similar a Django, con una comunidad algo más pequeña, y la última versión, 2.3.12, data del 6 de Abril de 2018
- web2py, es otro framework con todo incluido, desarrollo, debug, testing, acceso a base de datos..., con una comunidad parecida a TurboGears, pero con un problema principal, si bien es "Python3.x friendly", no tiene support oficial todavía para Python3.

Los frameworks no full stack más populares en python son:

- Flask, es un "microframework", en el cual puedes hacer todo lo normal que se haría en un full stack, pero no viene con muchas opciones por defecto. La intención viene dada por tener un servidor ligero, solo con lo necesario y nada más. La versión actual, 1.0.2, data de 2 de Mayo de 2018
- Bottle, similar a Flask, un microframework, pero está en una versión más temprana en su desarrollo, ya que la última versión estable es 0.12, del 13 de Diciembre de 2018
- CherryPy, un framework minimalista, según su propia descripción, es mucho más maduro en su última versión, 18.0, lanzada el 27 de Marzo del 2019. Aun así, sorprendetemente, su comunidad es relativamente pequeña.

Dada toda esta información, la decisión final estaba entre usar Django, dentro de los full stack el más prometedor, o Flask, su contra parte dentro de los no full stack. Dado que el objetivo de este trabajo es realizar la interfaz, sin tener que entrar en mucho detalle específico en cuanto a implementación, usar Flask y tener que implementar "tantas" cosas desde cero es un ejercicio excesivo para el ámbito del proyecto. Es por eso que *la decisión final ha sido usar Django*.

2.1.2. Django

Las principales características técnicas de Django son:

1. **Modelo Template View**, de forma similar al MVC tradicional mencionado previamente, Django funciona de forma ligeramente diferente. Las vistas en MVC son *templates*, es decir, plantillas. Estas plantillas pueden ser las que vienen por defecto en el framework, o se pueden utilizar las de otro motor, como puede ser Jinja2. Similarmente, el controlador en MVC corresponde con la View en Django, aunque aquí hay poca diferencia, más allá del nombre.

2. **Object Relational Mapper (ORM)**, es el middleware encargado de permitir la interacción con la base de datos que trae Django. Este ORM tiene varias ventajas, la primera, traduce código en sentencias SQL (por defecto Django tiene su ORM construido para SQL, aunque hay formas de conectarlo a bases de datos no relacionales), la segunda, y de gran importancia, este ORM se encarga de escapar las queries hechas a la base de datos, por lo que es una capa de seguridad contra SQL Injection que viene por defecto en Django. Por último, independientemente de la base de datos a la que esté conectado el servicio (por defecto vienen MySQL, PostgreSQL, SQLite, Oracle), la forma de hacer las queries es el mismo, con la ventaja añadida de PostgreSQL que tiene sus propios comandos para aprovechar sus ventajas.

3. **Panel de administración**, aunque para este trabajo no tenga utilidad, Django viene por defecto con un panel de administración en el que se pueden manejar directamente los objetos de la base de datos, crear, borrar... en función de lo establecido en las vistas

4. **Con pilas incluidas**, como se ha mencionado previamente, Django al ser un framework full stack, trae una serie de funcionalidades de serie que son de gran utilidad, y ayudan a reducir el tiempo de desarrollo. Por ejemplo, trae un modelo de usuarios en el que tiene incluido la opción de ser ”superusuario”, datos, cuándo fue la última vez que se registró en el sistema....
   Tiene un sistema de sesión para usuarios, un sistema de templates, y un módulo de seguridad que funciona en sincronía con el sistema de templates (se hablará de esto más adelante, en la parte de seguridad dentro de la sección de diseño y desarrollo)

5. **Modularidad**, otra de las ventajas que tiene Django es que funciona con una jerarquía. En la cúspide, está el proyecto como una entidad entera, (que comprende de aplicaciones, y configuración propia del proyecto), y

las aplicaciones, que se pueden crear ”al vuelo”, o importar. Esto permite reutilizar aplicaciones de otros proyectos, por ejemplo.

Actualmente Django es usado por una gran cantidad de empresas y servicios, siendo algunos de ellos: Mozilla Firefox, Youtube, DropBox, Google, NASA, Reddit, Instagram, Quora, BitBucket.

2.1.3. React

Existen varios frameworks para crear SPAs, algunos de los más conocidos son AngularJS, React, y VueJS. Debido a mi desconocimiento total previo a este trabajo en cualquiera de ellos, decidí usar el que tiene una comunidad más extensa, y en mi opinión el más sencillo de utilizar, React. Además, React es más ligero dentro de esta categoría de frameworks, ya que su principal uso es la representación visual, la vista en un modelo MVC.

React es una librería JavaScript creada y mantenida por Facebook. Puede ser usada para crear una SPA, o para aplicaciones móbiles, ya que está optimizado para representación de datos que pueden cambiar rápidamente.

React provee un lenguaje de templating y funciones *hook* para renderizar HTML. Mediante la creación de componentes que tienen un estado interno, es posible almacenar datos tales como el valor de un campo de texto, si se ha clickado en un elemento...

Ese estado puede ser usado para varios fines, puede ser pasado a componentes hijos, en lo que se denomina *props*, o puede ser usado para realizar llamadas al servidor enviando cierta información.

Las características principales de React son:

1. **JSX**
   React usa para la creación de componentes y representación visual JSX, JavaScript + XML. Este XML tiene un gran parecido con el HTML normal, y además permite crear *XML-like* tags propios, que serán los componentes. Aunque también se pueden crear elementos no XML, de una forma distinta, y algo menos intuitiva, como se puede ver en la figura 3
   (Por razones prácticas, ya que el XML, y los tags creados a partir de componentes van a renderizar a código HTML, a partir de ahora se definirá

como HTML, aunque lo correcto sería decir *código XML que en el cliente renderizará como código HTML*)

Figura 3: Diferencias entre JSX y non-JSX

2. **Components**

   Como se ha mencionado previamente, el bloque principal usado en en React se llama Component, el cual siempre tendrá una función *render*, que devolverá el código HTML a renderizar. Esta función *render* será llamada siempre que haya un cambio en su estado, en los props que se le hayan pasado al componente.

   Los componentes se dividen en dumb component, y smart component. Un dumb component se centra principalmente en la representación visual, por lo que está enfocado a ser reusable. Un smart component son los que tienen un estado interno, y se encargan de modifcar ese estado. También tienen representación visual, y son los que invocan a los dumb components pasandoles parámetros pertinentes. La diferencia entre un dumb y un smart component se pueden ver en las ejemplos de código 1, y 2. En el segundo, se puede ver cómo *onClickCustomDiv* cambia el estado, añadiendo 1 a *this.state.ntimes*, y al cambiar el estado, *render* será llamado de nuevo, por lo que lo que se visualizará será el nuevo valor

Código 1: Dumb component para un botón

```
export const Button = ({name, disabled, onClick}) => (
    <button
        className={ disabled ? name + " disabled_pagination" :
            name }
        disabled={disabled}
        onClick={onClick}
    />
);
```

Código 2: Smart component

```
        class CustomDiv extends Component {
          constructor(props){
              super(props)
              this.state = {
                  ntimes: 0,
              }
              this.onClickCustomDiv = this.onClickCustomDiv.bind(this);
          }
          onClickCustomDiv() {
              this.setState({ ntimes: this.state.ntimes + 1, })
          }

          render() {
              return <h1> Clicked {this.state.ntimes} </h1>
          }
      }
```

3. **Extensa cantidad de librerías**
   React es uno de los frameworks con más popularidad actualmente, y eso conlleva que el número de librerías (en este caso, paquetes NPM, Node Package Manager) para React es sumamente extensa, para cualquier componente existente, desde un botón, hasta un carrusel. En ocasiones, la oferta de paquetes es demasiado extensa, tener 6 o 7 paquetes del mismo tipo de componente puede ser algo abrumador, sobre todo si estás empezando en el framework, y no sabes cómo decidir

4. **Webpack**
   Típicamente, en el mundo de front siempre ha sido diferente, más ”sencillo” de desarrollar, en el sentido de no tener que estar compilando, no tener necesidad de aplicar principios de ingeniería de software... Con React es necesario realizar estas funciones.

5. **React DevTools**
   React tiene una extensión con este nombre, que sirve para debuggear de forma muy sencilla en el navegador. Esta extensión permite navegar en el DOM, e irá mostrando (cuando lo haya) el estado del componente. De esta forma se puede comprobar que el cambio de estado esperado esté occuriendo o no, por ejemplo.

Uno de los principales problemas que tiene React es que el estado de un componente solo se puede pasar a sus componentes hijos, pero no hay forma

de enviarlo al padre. Cuando existe esta necesidad, o la aplicación a desarrollar es relativamente compleja, se requiere de una tecnología adicional, Flux. Flux provee una store donde almacenar el estado de forma global, por lo que cualquier componente puede acceder a ello, pero es tedioso de implementar, ya que tiene una cantidad de *boilerplate* inicial enorme, uno de los mayores detrimentos de esta tecnología. Debido a que este proyecto no tiene esa necesidad, Flux no ha sido usado.

2.1.4. Base de datos

Una vez elegido el stack tecnológico para el *front* y el *back*, la última elección para la parte de las tecnologías web es la base de datos. Este trabajo no tiene un requerimiento especial en cuanto a la forma de guardar los objetos, y el modelo de datos no es complejo ni requiere de bases de datos no relacionales. Es por esto que la elección de una base de datos es ”trivial”, simplemente con elegir una base de datos relacional (SQL) es suficiente.

Se ha mencionado previamente que Django soporta nativamente cuatro bases de datos diferentes, Oracle, SQLite, MySQL, y PosgreSQL. De estas cuatro bases de datos, hay una en concreto, PostgreSQL, que provee una funcionalidad añadida, Full Text Search (FTS), provee la capabilidad de identificar ”documentos” (en este caso, documento se refiere a una unidad de búsqueda en un sistema de búsqueda Full Text Search) que satisfagan una query, y óptimamente, ordenarlos por relevancia con respecto a la query [PostgreSQL, 2019b].

Si bien en el resto de ellas se puede llegar a tener FTS, se ha elegido PostgreSQL ya que viene por defecto, sin necesidad de configuración adicional, y puede ser interesante estudiar la posibilidad de incorporar esta funcionalidad.

PostgreSQL

PostgreSQL es una base de datos relacional de código libre. El origen de esta base de datos data de 1986, como parte del proyecto POSTGRES, de la Universidad de California, del campus Berkeley. Esta base de datos (BBDD) se ha ganado una reputación por su arquitectura, *reliability*, set de funcionalidades robusto, extensibilidad, y la dedicación a la comunidad del software libre. Cumple con ACID (Atomicity, Consistency, Isolation, Durability, el set de propiedades de transaciones en una base de datos que la mantienen válida) desde 2001, y cuenta con potentes extensiones [PostgreSQL, 2019a].

Alguna de las muchas características de PosgreSQL:

- Data types
  Contiene una variedad de tipos de datos, desde los primitivos comunes, Integer, String, Boolean, Date, a tipos más complejos, Array, Documento JSON, XML, tipos geométricos: punto, linea, círculo y polígono, y personalizados.

- Integridad datos
  Similarmente a otras bases de datos, contiene UNIQUE, NOT NULL, claves primarias, foreigns, y también tiene exclusión por restricción

- Concurrency performance
  Contiene indexación "simple" (B-Tree, multicolumna, parcial, expresiones), avanzada, mediante árboles o índices invertidos..., paralelización de queries de lectura y construcción de B-trees index, e incluso comilación Just-In-Time de expresiones

- Internacionalización y Full Text Search
  Soporta set de caracteres internacionales, y tiene como funcionalidad Full Text Search.

2.2. Machine Learning

Machine learning es la ciencia (y arte) de programar computadores para que puedan aprender de ejemplos [Géron, 2017]

Como se ha comentado previamente, en este trabajo se utilizará Machine Learning para extraer el texto en imágenes con texto manuscrito. No obstante, en primer lugar, se hará una descripción breve sobre los fundamentos de las técnicas que se engloban en este campo.

Machine Learning (ML) es un subcampo de la Inteligencia Artificial (IA). La IA es un campo muy amplio que abarca desde un problema de clasificación por árboles, hasta filtros de spam en el correo, pasando por sistemas de recomendación de productos. Hay varios tipos de sistemas de ML, y su clasificación depende de:

- Si están o no entrenados con ejemplos "etiquetados", es decir, con ejemplos para los cuales se conoce previamente la clase o categoría a la que pertenecen, o si requieren la interacción con el entorno para aprender (supervisados, no supervisados, semi supervisados, y Aprendizaje por Refuerzo).

- Si pueden aprender *al vuelo* (aprendizaje online vs batch).
- Si funcionan comparando nuevos datos con datos conocidos, o en su lugar, detectan patrones en los datos de entrenamiento, y construyen un modelo predictivo (aprendizaje instance-based vs model-based)

Esta diferenciación no es única y, de hecho, pueden ser combinados de diferentes formas, por ejemplo, un bot que aprenda a jugar al Risk con una red neuronal mediante aprendizaje por refuerzo podría ser un modelo online y model-based.

A continuación, se hará una breve introdución de los problemas que pueden surgir en Machine Learning (Sección 2.2.1), qué significan las formas de clasificar (Sección 2.2.2), y por último, pasar a hablar más a fondo de redes neuronales (Sección 2.2.3).

2.2.1. Problemas en Machine Learning

Principalmente pueden existir dos problemas principales, pero cada uno de ellos engloba varios subproblemas en si mismos.

Mala elección de algoritmo En este caso el problema es ”sencillo” de solucionar, no todos los algoritmos sirven para el mismo problema. Para un sistema de predicción temporal (por ejemplo, predicción meteorólogica), no se puede aplicar clustering, ya que simplemente no va a funcionar nunca, o si funciona es puro azar. Scikit Learn (librería de Python de Machine Learning, Computer Vision y computación científica) nos presenta con la figura 4 [2], en la que se sugieren algoritmos (aunque no contengan todos los existentes, ni sus variaciones, es una buena guía) en función de la cantidad de datos, y la tarea a realizar.

Referido a datos

1. **Cantidad insuficiente**, si bien para los humanos es relativamente fácil aprender patrones con muy pocos ejemplos, es el caso contrario para la mayoría de algoritmos de ML, necesitando miles, o incluso millones de ejemplos. Una investigación llevada a cabo por Microsoft [Banko and Brill, 2001] reveló que diferentes técnicas de variada complejidad, entrenadas para la misma tarea, tendían a converger a un valor de aceptación común dado un número suficientemente grande (en las centenas y miles de millones), pero no siempre es posible disponer de tal cantidad.

[2] Extraída de Scikit Learn [Scikit-Learn, 2019]

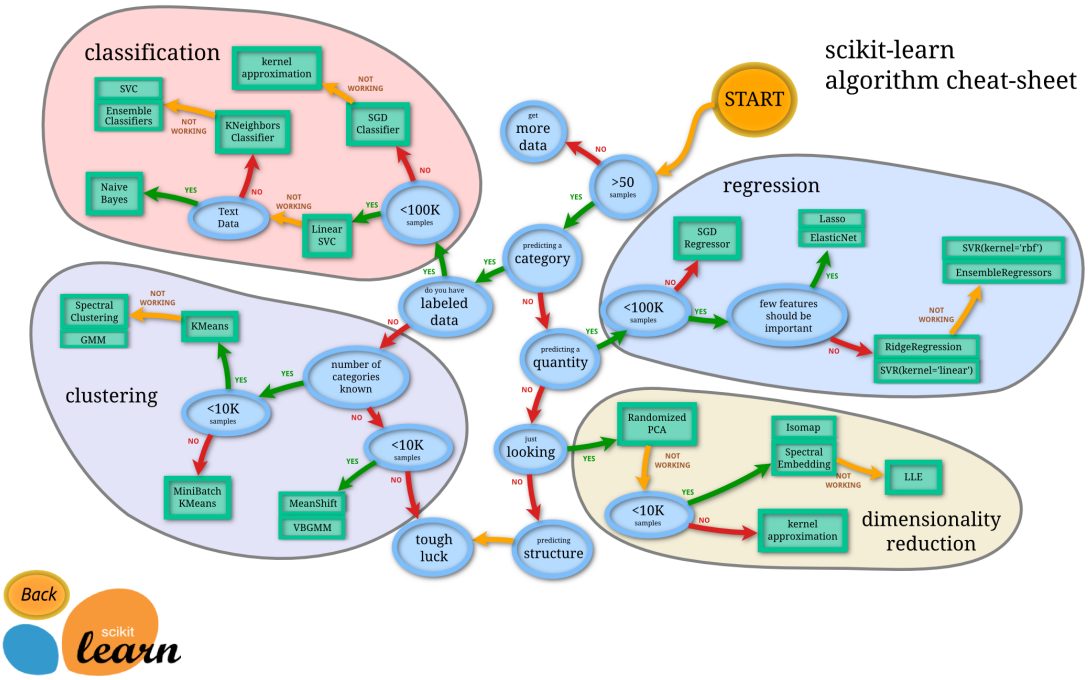

Figura 4: Sugerencia de algoritmos a utilizar en función de la tarea

2. **Datos no representativos**, el set de datos debe tener contenido que represente la realidad que se quiera predecir. Por ejemplo, se quiere entrenar un modelo que dados datos meteorólogicos y de tráfico, prediga la polución futura en las ciudades de Asia, por lo que en este modelo no tendría cabida los datos de tráfico de pueblos, ni de ciudades que no estén en Asia.

3. **Calidad de ejemplos**, siguiendo el ejemplo anterior del predictor de polución, si, una vez se tengan solo datos de ciudades en Asia, dichos datos tienen muchos campos vacíos, o se añaden regiones inhóspitas al set de datos, van a añadir ruido al modelo, y no predecirá con la misma precisión.

4. **Rasgos irrelevantes**, si al set de datos de tráfico en ciudadesy meteorología, le añadimos las veces que el emperador de Japón ha pisado la ciudad, no va a tener relevancia. Lo mismo con el número de coches grises, no es algo que aporte valor al modelo.

5. ***Overfitting***, término que corresponde con ”sobreentrenar” el modelo para que se ajuste a los datos de entrenamiento. Cuando se sobreentrena, o mejor dicho, cuando ocurre el overfitting, el modelo no es capaz de generalizar más allá de los datos proporcionados, por lo que un ejemplo que varíe algo, no va a ser predecido de forma correcta.

6. ***Underfitting***, el caso contrario al de Overfitting, cuando el modelo no ha

sido entrenado lo suficiente, es demasiado general, y no predecirá de forma correcta para el problema que esté siendo entrenado el modelo.

Testing y validar Aunque no sean problemas en sí que ocurren con los datos de ejemplo, es algo a tener en cuenta en Machine Learning, y que puede ocasionar problemas. Los ejemplos a la hora de entrenar se suelen dividir en dos sets, el set de entrenamiento, y el set de testing. Se entrena el modelo con el primer set, y se comprueba la precisión con el segundo.
Pero esto a veces no es suficiente, por lo que el set de entrenamiento se divide (dependiendo de la técnica) en dos, entrenamiento y validación. El problema puede venir si no se elige un test de testing, o si la forma de separar entrenamiento y validación no es lo suficientemente buena.

2.2.2. Clasificación de modelos de Machine Learning

La clasificación más habitual dentro de ML es la que separa el tipo de aprendizaje en aprendizaje supervisado y no supervisado. A continuación se describe en qué consiste cada uno de ellos:

- **Supervisado**, en este tipo de modelos, los datos que se le entregan al modelo entrenado incluyen la solución (a veces denominado clase). Un par de ejemplos de tareas típicas de este tipo de modelos son la clasificación, y la regresión.
  Algunos de los algoritmos más importantes son: k-nearest vecinos, regresión linear, regresión logística, Support Vector Machines (SVM), árboles de decisión, y redes neuronales.

- **No supervisado**, el contrario que el supervisado, los datos carecen de clase, y se le pide al modelo que infiera las clases a partir de los datos.
  Las tareas, y algoritmos asociados, que pertenecen a este tipo son:

  - Clustering (k-Means, Hierarchical Cluster Analysis HCA)
  - Visualización y reducción de dimensionalidad (Principal componente Analysis PCA, Kernel PCA...)
  - Detección de anomalías
  - Reglas de asociación (Apriori, Eclat)

- **Semi supervisado**, algoritmos que pueden tratar con datos parcialmente etiquetados con clases, normalmente con mayoría de datos sin clase. Clustering, Deep Belief Networks (DBN) son ejemplos de usos de este modelo.

- **Aprendizaje por Refuerzo**, bastante diferente a los otros tres tipos, funciona con un sistema de aprendizaje mediance un *agente* que puede observar el sistema, selecionar y realizar acciones, y obtener recompensas de esas acciones (pueden ser recompensas negativas). Debe aprender cuál es la mejor política de comportamiento, para obtener la mayor recompensa a lo largo del tiempo. La política define qué acción realizar en función de la situación [Sutton and Barto, 2018]

En cuanto a las formas de aprender, se distinguen dos tipos:

- **Batch**, este sistema es incapaz de aprender de forma incremental, hay que entrenarlo con todo los datos disponibles. Por lo general este proceso tarda tiempo, y se hace offline, primero el sistema se entrena, y lanza a producción, donde es ejecutado sin necesidad de volver a entrenar.
  Si se quiere entrenar con nuevos datos (o adicionales), hay que re-entrenar el sistema entero desde cero (excepto en Transfer Learning, se hablará de esto en la sección de Neural Networks). Este modelo tiene como ventaja disponer de las predicciones desde el momento en el que se entrena, pero entrenar puede ser costoso (tanto en tiempo como dinero), y puede incluso llegar a ser no viable si la cantidad de datos es demasiado grande, o la unidad que lo procese (CPU/GPU) no es lo suficientemente potente.

- **Online**, se entrena el sistema incrementalmente, alimentando el modelo secuencialmente, o bien datos individuales, o bien en grupos pequeños llamados mini-batches. Entrenar es ”barato”, debido a la poca cantidad de datos, por lo que se puede aprender al vuelo. Esta forma de aprender es adecuada para sistemas en los que los datos llegan de forma continua, por ejemplo, precios de bolsa, y es necesario adaptar de forma rápida y/o autónoma.
  Algo a tener en cuenta en este tipo de modelos es el *learning rate*, (ritmo de aprendizaje), un parámetro que configura la velocidad a la que se adapta el modelo a los nuevos datos, un rate alto significa que se adaptará rápido, pero olvidará también rápido datos antiguos, y al contrario con un rate bajo.

En cuando a las formas de generalizar, se distinguen a su vez dos tipos:

- **Instance-based**, un sistema que aprende de ejemplos ”de cabeza”, y usa esos ejemplos para generalizar nuevos casos usando una medida de similaridad.

- **Model-based** construye un modelo de los ejemplos, y usa ese modelo para hacer predicciones.

2.2.3. Neural Networks (NN)

Las redes de neuronas artificiales son un intento de imitar las neuronas del cerebro humano, ya que es una de las máquinas de computación más potentes que existen. Las redes de Neuronas artificiales (NN) entran dentro de técnica Machine Learning, por lo que todo lo mencionado hasta ahora en esta sección es aplicable, una NN es un intento de generalizar datos de la forma más precisa posible, con los posibles problemas que puede tener, al elegir el tipo de red, y la forma de entrenarlo (hiperparámetros).

El origen de las NN surgió en 1943 [McCulloch and Pitts, 1943], donde se definió por primera vez lo que se denomina perceptrón, una red con una sola neurona, la figura 5 es una representación visual de dicha neurona.

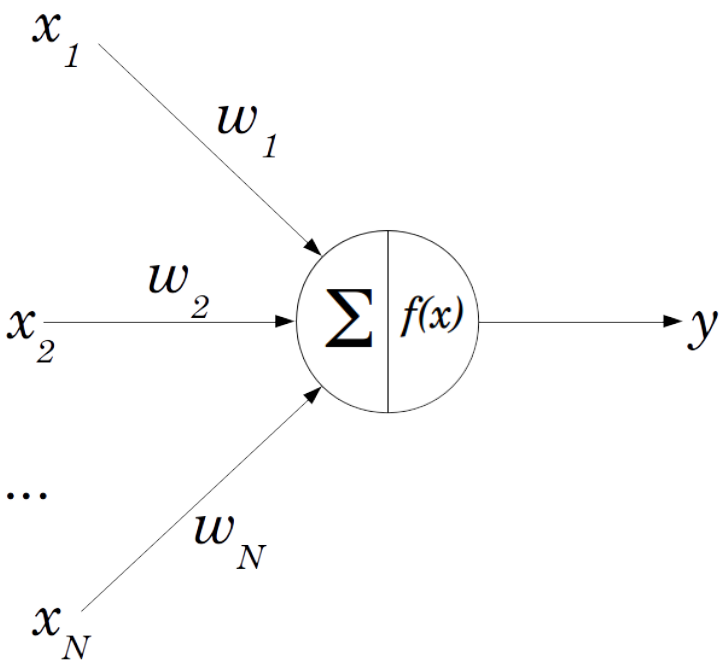

Figura 5: Perceptrón

Como se puede observar, y esto es aplicable en general a redes con más neuronas, cada neurona se compone de tres partes, los inputs $(x_1, ..., x_N)$, cada uno de ellos tiene un peso determinado $(w_1, ..., w_N)$, la neurona en sí, que consiste de una función de activación, y el output, que es el resultado de la función (dependiendo del tipo de función usada, dado un valor se activa). Adicionalmente, pueden tener un valor de input "especial", llamado *bias*, que añade $+1$ a la función de activación. Este valor de bias supone una translación en la función de activación que puede ayudar a encajar mejor el modelo para predecir mejor la realidad.

Para calcular el valor de la función, se calcula el sumatorio de los inputs ponderados con el peso, junto con el valor de bias w_b

$$y = w_1 x_1 + ... + w_M x_N + w_b = \sum_{\mathtt{i}=1}^{N} w_{\mathtt{i}} x_{\mathtt{i}} + w_b$$

Una red de neuronas trata de detectar patrones, pero hace falta encontrar los parámetros que detectean esos patrones correctamente. Para entrenar una red de neuronas, se inicia con pesos aleatorios, y se le presentan ejemplos que se asemejan a aquello que se quiere predecir o detectar. En cada paso del entrenamiento se analiza el resultado de los pesos contra el ejemplo deseado, y se modifican. Si el error calculado en el paso disminuye, se sigue modificando en la misma dirección, pero si el error aumenta, se modifica. La forma de calcular este proceso es con *Gradient descent*, (representación gráfica en la figura 6), [3] que mide el cambio en los pesos respecto al cambio en el error. Ya que este diferencial de cambio se calcula con una derivada, se puede ”observar” visualmente como la pendiente de una función. Cuanto mayor es el ratio de aprendizaje, mayor será el gradiente, por lo que existe el riesgo de ir demasiado lejos, y no encontrar el mínimo deseado, pero un ratio pequeño puede ralentizar demasiado el proceso de aprendizaje, es por esto que algunos sistemas, como por ejemplo *fastai* permite utilizar un ratio de aprendizaje variable [FastAI, 2019]

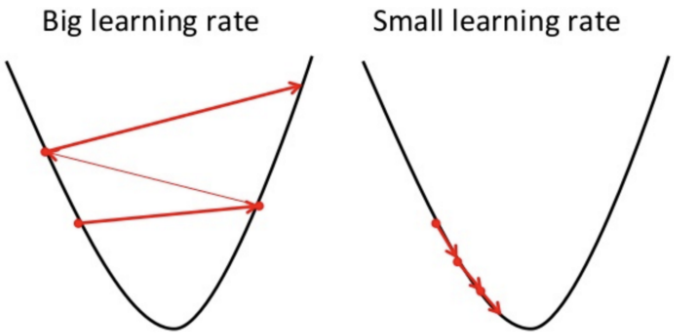

Figura 6: Gradient descent y su relación con el ratio de aprendizaje.

Hay muchos tipos de redes neuronales, para usos muy variados, se pasa a detallar los tipos más relevantes para este proyecto, pero eso no quiere decir que sean los únicos tipos.

[3] Imagen obtenida de https://towardsdatascience.com/gradient-descent-in-a-nutshell-eaf8c18212f0

Tipos de redes de neuronas

- **Multilayer Perceptron (MLP)**
  Cuando una red tiene varias capas, con tres tipos definidos, input (entrada) layer, hidden (ocultas) layers, y output (salida) layers, y cada neurona en cada capa está conectada a todas las neuronas de la capa siguiente, se le denomina Multilayer. Adicionalmente, se suele utilizar un algoritmo llamado backpropagation, mediante el cual después de actualizar desde la primera capa (input) hsata la última (output), se hace una propagación en dirección contraria, desde el output hasta el input. La figura 7 [4]

  muestra un modelo MLP, sin backpropagation.

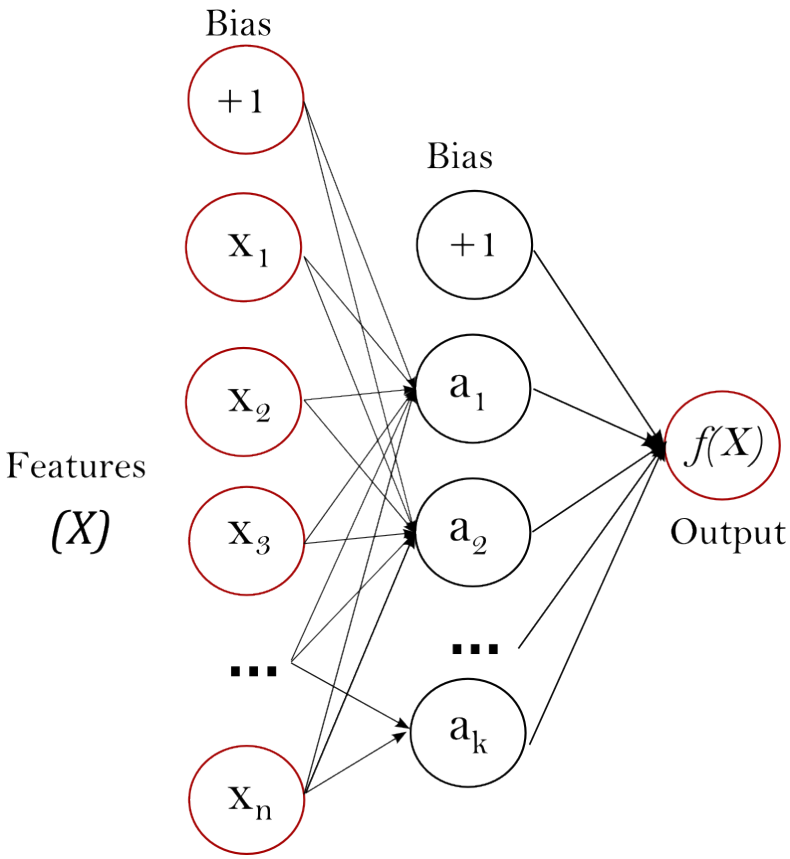

Figura 7: MultiLayer Perceptron

- **Convolutional Neural Network (CNN)**
  Este tipo de red, usada principalmente para imágenes, es una variación de una MLP, tiene una capa de inputs, otra de outputs, tiene una o más hidden capas, y adicionalmente tiene capas de convolución. En estas capas, se definen filtros en cada capa que son capaces de detectar patrones en las imágenes. En

[4] Extraída de $https://scikit-learn.org/stable/modules/neural_networks_supervised.html$

las capas tempranas de la red los filtros solo son capaces de detectar patrones sencillos, formas geométricas, bordes... pero en capas más profundas dentro de la red los patrones que se pueden detectar son más avanzados, pudiendo detectar formas completas, ojos, pelo, plumas, y en las últimas capas de convolución pueden detectar objetos enteros, perros, gatos, una cara...
Un filtro es una matriz de tamaño NxM, con el cual se calcula el producto escalar con cada submatriz MxN de la representación de la imagen.

- **Recurrent Neural Network (RNN)**
  Las conexiones entre nodos forman un grafo dirigido a lo largo de una secuencia temporal. Se diferencian de otros tipos de redes porque pueden usar el estado interno ("memoria") de la red para procesar secuencias de inputs, en vez de un input unitario. Esto las hace aplicables para tareas como reconocimiento de texto manuscrito sin segmentar, o reconocimiento de voz.

2.2.4. Reconocimiento de texto manuscrito

En esta sección se hace una descripción del problema en sí, reconocer texto manuscrito, a la par que se explica el estado del arte.

El dominio del reconocimiento de texto escrito a mano tiene dos variantes, la versión *online* y la versión *offline* [Tappert et al., 1990]. La versión online implica reconocimiento en tiempo real de una palabra o caracter. Entra dentro de este subdominio por ejemplo, las tablets (tanto las versiones antiguas, por ejemplo las Wacom, como las versiones nuevas), en las cuales se puede instalar programas que reconozcan una palabra según se escribe en la pantalla. Para esta tarea, reconocida comunmente como más sencilla que su contraparte, se tiene en cuenta, para un momento temporal t_n, tanto la trayectoria del pen (o dedo) en ese tiempo t_n, como el trazo pintado desde $t = 0$ hasta $t = t_n$.
La segunda versión, la offline, implica el caso contrario, una vez se tiene el texto (puede ser un párrafo, una palabra, o un caracter), analizarlo e intentar predecir. Esta versión es la que incumbe a este proyecto, ya que es la necesidad existente. La diferencia no reside únicamente en si el reconocimiento se hace en tiempo real, o si se dispone de menos datos (ya que con offline no se dispone de la trayectoria del trazo), en esta versión el problema se multiplica si se lleva al dominio de reconocimiento de texto, en vez de quedarnos simplemente en clasificar una palabra o caracter.

Una imagen general del problema se sitúa entonces en las siguientes subtareas:

- Encontrar el texto en la imagen, y extraer la parte (o partes) de la imagen concreta que contienen el texto
- Segmentar las líneas del texto, si bien el texto a máquina es sencillo, ya que las separaciones serán siempre las mismas, el texto escrito a mano no siempre es igual de preciso
- Segmentación de la línea en palabras
- Reconocer la palabra. Esta última parte del problema ha sido enfocado de dos formas diferentes a lo largo de la historia, uno de esos enfoques es segmentar la palabra de nuevo, con la intención de reconocer caracteres individuales, y el segundo enfoque es no segmentar la palabra, tratarla como un ente único, e intentar reconocer los caracteres dado el contexto de cada caracter con respecto a la palabra

Uno de los intentos tempranos de detección de texto envuelve el uso de múltiples clasificadores (Bayes, K-NN, y un clasificador basado en distancia, pero no especifican cuál exactamente) entrenados separadamente, y la creación de varios sistemas de voto entre ellos, para hacer reconocimiento de caracteres [Xu et al., 1992]. Aunque logran una precisión alta, entorno al 94 %, admiten que necesitan más ejemplos para que el valor de precisión sea empírico.

Similarmente, un estudio [Tang et al., 1998] sobre reconocimiento de caracteres chinos (que por naturaleza es diferente del alfabeto latín, y de otros alfabetos, como puede ser el japonés o el árabe), usan extracción de características (usando tres, *forma periférica*, *densidad de trazos*, y *dirección de trazo*) junto con un clustering creado por ellos mismos, (en el que un caracter puede tener características en diferentes nodos a la vez), para reconocer caracteres chinos, pero no llega a ser todavía el resultado deseado.

Más tarde, un estudio [Plamondon and Srihari, 2000] hace un repaso al estado del arte en el momento, y es uno de los primeros exponentes de segmentación de líneas, que es una técnica que se tiene que llevar a cabo en presencia de un párrafo. Dado una imagen de un párrafo, hay que separar el párrafo en líneas, lo cual es sencillo si el texto es a máquina, pero no tan sencillo con texto a mano, para poder realizar reconocimiento de palabras individuales en cada linea. Para cada palabra hay dos tipos de análisis, holístico, en el cual se decide en función de la palabra

entera, o analítico, mediante el cual se reconoce cada caracter de forma individual.

El enfoque común (en el momento) era usar Hidden Markov Models (un Markov Model es un modelo estocástico usado para modelar sistemas que cambian de forma aleatoria, basado en una secuencia de estados. El tipo HMM se usa cuando el sistema es autónomo, y el estado del dicho sistema es parcialmente observable, por ejemplo, reconocimiento de voz, tiene como estado observable la onda del audio del sonido, y como estado oculto el texto hablado) para estructurar el proceso de reconocimiento, un lexicon pequeño del que se deriva uno más grande. Admiten en el artículo que a pesar de ser un buen avance, en casos concretos el resultado es bueno, pero en general faltaba por investigar. No es hasta el año 2009 cuando se registran dos resultados prometedores.

El primero [Plötz and Fink, 2009] muestra un modelo que usa HMM y Modelos encadenados de Markov (del inglés Markov Chains) en una serie de pasos que se mencionan a continuación,

1. Dada una imagen, realizar detección de texto, es decir, localizar dónde hay texto, sin identificar o reconocer ninguna palabra
2. Una vez detectado el texto, extraer las líneas de texto
3. Por cada línea de texto, llevar a cabo los pasos *baseline correction*, y *normalization*, que sirven para corregir defectos en las lineas detectadas.
4. Serialization and feature extraction. Ya que las HMM son secuenciales, tienen que tener un orden. Para representar este orden se usa típicamente una *sliding window*, que es un frame de N x M que recorre una imagen, visitando un recuadro equivalente a ese tamaño en cada paso. Después de realizar serialization, en cada ”tira” de píxeles generada por la sliding window, se realiza feature extraction en esa ”tira”.
5. Por último, se envían los datos al *Model decoding*, que contiene a su vez dos componentes:
   - Model de escritura, compuesto por un HMM, con finalizad de extraer características e identificar caracteres
   - Markov Chain. Provee modelo probabilístico para dependencias a largo plazo.
6. Por último, se comprueba la hipótesis dada por el Model encoding

Para los resultados usan la métrica Word Error Rate (WER), cuya fórmula es como sigue

$$WER = \frac{S + D + I}{S + D + C}$$

Donde S es el número de palabras substituidas, D las borradas, I las introducidas, y C las palabras correctas. Esta fórmula es derivada de la distancia de Levenshtein, también llamada *edit distance*, ya que provee del número de cambios necesarios para llegar de una palabra a otra. Es por esto que a mayor WER, mayor error en el modelo. Los resultados finales de este artículo listan para casos de datasets "generales" (IamDB, por ejemplo, en el que los ejemplos vienen de todo tipo de escenarios) un WER entre el 10 % y el 40 %, y para datasets específicos (como puede ser un dataset que contiene códigos postales de Estados Unidos escritos a mano, entre otros) el WER tiene un valor entre 0.8 % y 12 %. Estos resultados, aunque no perfectos, son un gran cambio con respecto a previos.

El segundo resultado prometedor viene de la mano de Graves, en su artículo [Graves and Schmidhuber, 2009], uno de los primeros en usar redes de neuronas, con el auge de esta tecnología entorno a estos años. Graves defiende que uno de los problemas de los enfoques que usan HMM es que las características que extraen estos modelos tienen que ser diseñadas de forma muy específica por cada alfabeto, y por cada idioma. En su solución, crea un modelo capaz de identificar árabe, sin tener él, ni su co-autor, una iota de árabe.

El modelo que crean se basa en tres componentes:

- Multi-Dimendional Recurrent Neural Network(MDRNN). Como se ha mencionado antes, una RNN es un tipo de NN que forma una temporalidad, y puede guardar estado, formando una "memoria" interna de la red. En este artículo usan una red RNN que trata con una dimensionalidad mayor a la habitual (pasan de vectores de una dimensión a dos dimensiones), y específicamente, usan una versión de este tipo de redes que se llama Long Short-Term Memory (LSTM). Esta versión de RNN consiste de "células de memoria" conectadas recurrentemente, cada una de las cuales tiene tres puertas multiplicativas, input y output (como el resto de las redes), y adicionalmente, otra entrada que es *forget* (olvidar), con la que se puede modelar cuándo una célula debe olvidar el contenido.
  En este artículo, cuando se procesa cada píxel de la imagen, recibe de capas ocultas el contexto del píxel, es decir, la información de los 4 píxeles adyacentes. Para realizar esto, hay una capa oculta por cada dirección desde

la que se quiere dar el contexto.

- Connectionist Temporal Classification. Es una capa output para etiquetar secuencias con RNNs. No requiere de datos presegmentados, entrena la red a proporcionar probabilidades condicionales en las posibles etiquetas dada una secuencia de input. Esta capa contiene una unidad más que elementos en el alfabeto $\mathcal{L}$ de etiquetas, donde los primeros $|\mathcal{L}|$ outputs estiman la probabilidad de observar la etiqueta correspondiente, y las probabilidades extra son el espacio, y la ausencia de etiqueta.

- Network hierarchy. El último "componente" es la forma de realizar la jerarquía de la red. En este artículo definen una serie de pasos cada vez que se trata una imagen, para reducir la dicha imagen. Estos pasos se repiten tantas veces como sea necesarios,

  1. Dada una imagen de una palabra, o frase, dividirla en bloques
  2. Las cuatro capas ocultas de la MDLSTM escanean en cada dirección (derecha-abajo, derecha-arriba, izquierda-abajo, izquierda-arriba)
  3. Las activaciones de cada capa oculta se coleccionan en bloques
  4. Estos bloques se dan como input a una capa feedforward para que los procese

Este artículo en concreto habla de la creación de un modelo de reconocimiento de texto *árabe*, sin saber los autores del artículo nada de árabe. Los resultados obtenidos son muy buenos, llegando a ganar la competición *ICDAR 2007 Arabic handwriting recognition contest*, teniendo entre un 91.43 % y un 96.75 % de precisión (en dos modalidades diferentes de la competición).

Por último, caben mencionar dos trabajos más hechos al respecto, el primero, por un uso de una tecnología totalmente diferente hasta las mencionadas hasta ahora, Algoritmos Genéticos (GA). En este artículo, [Rahul Kala and Tiwari, 2010], se genera un subgrafo por cada letra del alfabeto a tratar, para juntarlo todo en un grafo conjunto, y se "camina" por el grafo dada información extraida de una imagen que tiene texto. Aunque los resultados son excelentes, con un 98.44 % de precisión, el sistema está entrenado con muy pocos datos (385 inputs en total de 69 caracteres), por lo que puede dar lugar a overfitting, es decir, que el sistema no esté lo suficientemente generalizado.
El segundo, algo más reciente, habla de combinar HMM preconfigurado para caracteres concretos con LSTM para reconocimiento de texto. Es por lo tanto una combinación de técnicas anteriormente vistas, que consigue una mejora relativa

del 5.6 % (en WER) respecto a previos trabajos [Doetsch et al., 2014].

Como conclusión, se puede decir que los métodos que funcionan de forma suficientemente buena a día de hoy vienen dados por el uso de Deep Learning (término usado generalmente para redes neuronales con capas ocultas), en concreto usando CNN para el tratamiento de imágenes y reconocimiento de patrones visuales, y RNN para la contextualización de caracteres, y por otro lado, HMM a priori parecen ser útiles tanto como método alternativo, aunque no sean tan precisos, o como soporte de ayuda a las redes neuronales.

En este sentido, la API proporcionada por Google, API Vision [Google, 2019], se está convirtiendo en una de las herramientas punteras para el reconocimiento de texto tanto manuscrito como no. Aunque Google no desvela en ningún sitio qué tipo de red neuronal (o combinación de ellas) usa, es razonable decir que usa este tipo de tecnologías basadas en Deep Learning.

Por otro lado, si bien Microsoft Azure hace uso de las mismas tecnologías [Microsoft, 2018], con Faster R-CNN [Ren et al., 2015], aplican una técnica llamada transfer learning. Esta técnica [Pan and Yang, 2010] se basa en usar una NN pre-entrenada y se entrena de nuevo, pero sólo se recalculan los pesos de la última capa. Esto permite ahorrar mucho tiempo y capacidad de computación

2.3. Bibliotecas digitales

En esta sección, vamos a repasar algunos de los portales existentes de textos manuscritos. Esta tarea ha sido llevada a cabo por bibliotecas, con algunos casos de asociaciones de estudiantes que tienen apuntes. Algunos de los ejemplos (previamente mencionados) son [MP, 2019, BNE, 2019, CSIC, 2017, BC, 2019, La-Bisagra, 2018, BiblioFCC, 2019]. Se ve que muchos no permiten buscar por el texto dentro de las imágenes. Este es el caso de la Biblioteca Nacional (Figura 8), donde se puede buscar por los campos típicos de autor, título, fecha, pero no por texto dentro de los documentos.

Sólo [BC, 2019] permite buscar por texto, pero no indica en qué páginas se encuentra el resultado. Google Books es otro de los pocos recursos que permite buscar por contenido dentro de un documento, siendo las opciones libros, revistas o periódicos, pero contiene una parte limitada del documento, no todo el documento es accesible sin pagar. Además, hasta donde sabemos, no contiene documentos manuscritos, que es el eje central de este proyecto.

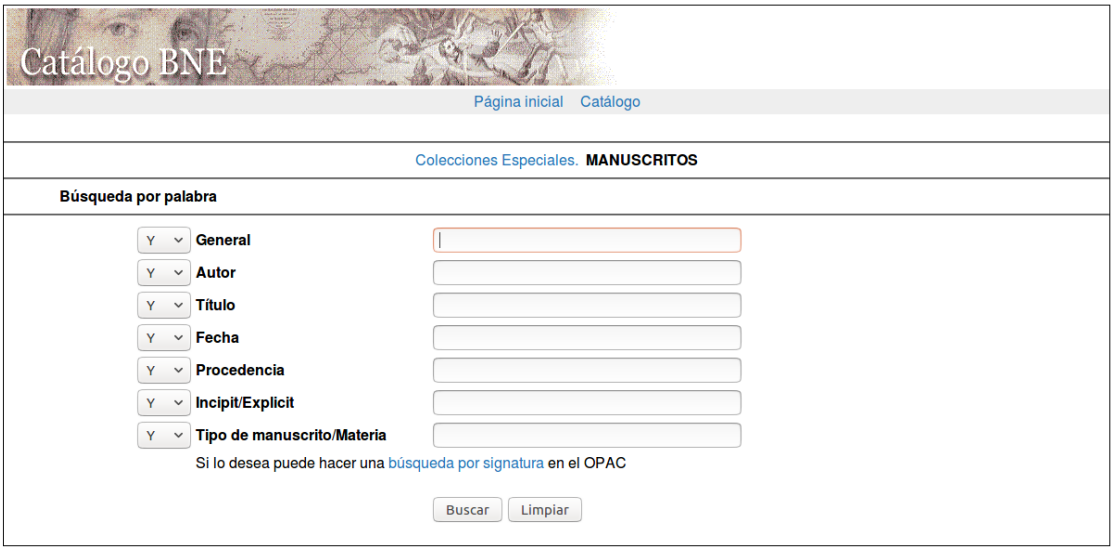

Figura 8: Buscador de manuscritos de la Biblioteca Nacional

3. Descripción del sistema

En esta sección, se hará un análisis del sistema a desarrollar. Para ello, se describirán en primer lugar los casos de uso, y después los requisitos tanto funcionales como no funcionales. Después se presentará el diseño de la arquitectura del sistema, con una explicación detallada de cada parte del mismo.

3.1. Introducción

La finalidad del trabajo es crear una interfaz que permita a usuarios buscar por texto en imágenes que tengan texto manuscrito. Para ello habrá que diseñar varios componentes: un portal web que sirve como interfaz, un servidor en el que se base el portal web, un servicio de Machine Learning para la detección automática de texto manuscrito, y por último una base de datos en la que se sustente el servidor.

Hasta hace unos pocos años, todas las páginas web eran servicios monolíticos en los que un servidor hacía las veces de front (cliente) y de back (servidor), con el patrón de diseño comunmente conocido como Modelo Vista Controlador (MVC), donde la "Vista" corresponde a lo comunmente definido como *front*, lo que se le presenta al usuario, y el controlador el *back*, la parte servidor y el modelo, con la base de datos (o sistema de persistencia utilizado) Recientemente, nuevas tecnologías han hecho posible desacoplar ambos componentes, en una aplicación web (front), y una web API (Application Programming Interface), se hablará en más detalle de ambos componentes más adelante. Este desacoplamiento

proporciona varias ventajas, entre ellas:

- Modularidad, tener componentes separados permite diferente ritmo de desarrollo, no depender tanto el uno del otro...
- Recursos/expertos, elegir tecnologías para crear una interfaz web puede tener sus limitaciones si ambos componentes **tienen** que usar el mismo lenguaje de programación, mientras que tenerlos por separado te permite más flexibilidad a la hora de contratar
- Builds/Deployment, cuando tienes un sistema monolítico, y haces una actualización de alguna parte del sistema, tienes que hacer un *deploy* de todo el sistema completo, mientras que con una división, puedes hacer diferentes deploys en función de las necesidad. Adicionalmente, permite manejar versiones de forma independiente
- Adaptabilidad, si por alguna razón, hay una necesidad de cambiar una parte del código de tu sistema, no hay que rehacer todo desde cero, hay una mayor adaptabilidad al poder cambiar todo tu *back* de PHP a Ruby, por ejemplo, mientras que el *front* se mantiene en React sin ser afectado
- Escalabilidad, en sistemas desacoplados, hacer un sistema escalable es más sencillo, puesto que solo tienes que hacer escalable el servidor, y no la aplicación web (esta se encarga solo de actuar a modo de interfaz). Por poner un ejemplo, en este tipo de sistemas, la autentificación de usuarios pasa de ser por sesión a ser por JSON Web Token (JWT), o por simple token (se hablará más adelante de los diferentes tipos de autentificación y el usado en este sistema)

Pero no todo son ventajas, algunas de las desventajas que tienen este tipo de sistemas son

- Algunos *frameworks* han sido construidos con propósito de server a modo de MVC, por lo que se pueden perder algunas funcionalidades
- Dos componentes distintos significa dos deployments distintos, tests unitarios individuales, además de tests conjuntos para comprobar que la conexión back-front es correcta...

Aún con todo, las ventajas de un sistema desacoplado superan a las de un sistema monolítico, y es por esa razón por la que para este proyecto se ha decidido hacerlo con un sistema desacoplado, teniendo en el servicio front Node con React,

y en el back Python con Django (haciendo uso de Django-Rest-Framework). Tras la especificación de los casos de uso y requisitos, en cada de las secciones individuales se profundizará más en estas tecnologías y el razonamiento de su elección.

En cuanto a la base de datos, siguiendo en la línea de escalabilidad y desacoplamiento, se ha usado ElephantSQL, un servidor PostgreSQL en la nube. Esto da varias ventajas, la primera, este servicio es un *plug-and-play*, permite montar un servidor BBDD con PostgreSQL en menos de 10 minutos, lo cual ayuda a acelerar el proceso de desarrollo. Además, permite liberar recursos en el servidor web, puesto que no tiene que encargarse de todas las operaciones CRUD que recibe, simplemente de hacer una petición a este servidor, y recibir el resultado, dejando así disponibles recursos para atender mientras tanto otras peticiones.

Por último, para detectar texto manuscrito en imágenes, se ha usado Google Vision API. Al ser una API, de nuevo ayuda a escalar el sistema, puesto que, al igual que el servidor PostgreSQL, el servidor web no tiene que realizar ningún cálculo ni detección, solo tiene que realizar una petición, y recibir la información. Se discutirań los detalles relacionados a Machine learning más adelante.

3.2. Especificación de los casos de uso

En las figuras 10 y 9 se muestran una representación gráfica de los casos de uso y de los diferentes actores que pueden interactuar con el sistema.

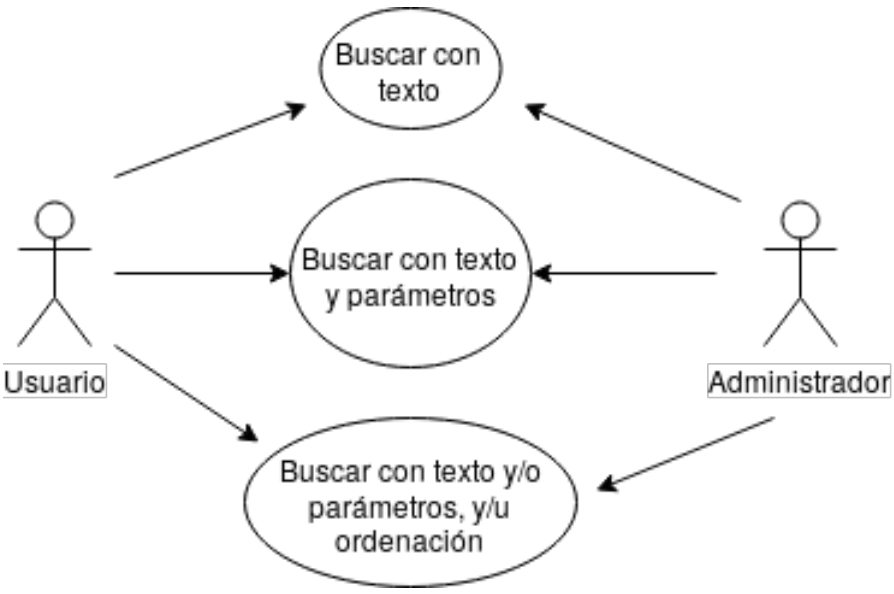

Figura 9: Diagrama de Casos de Uso para usuario y administrador

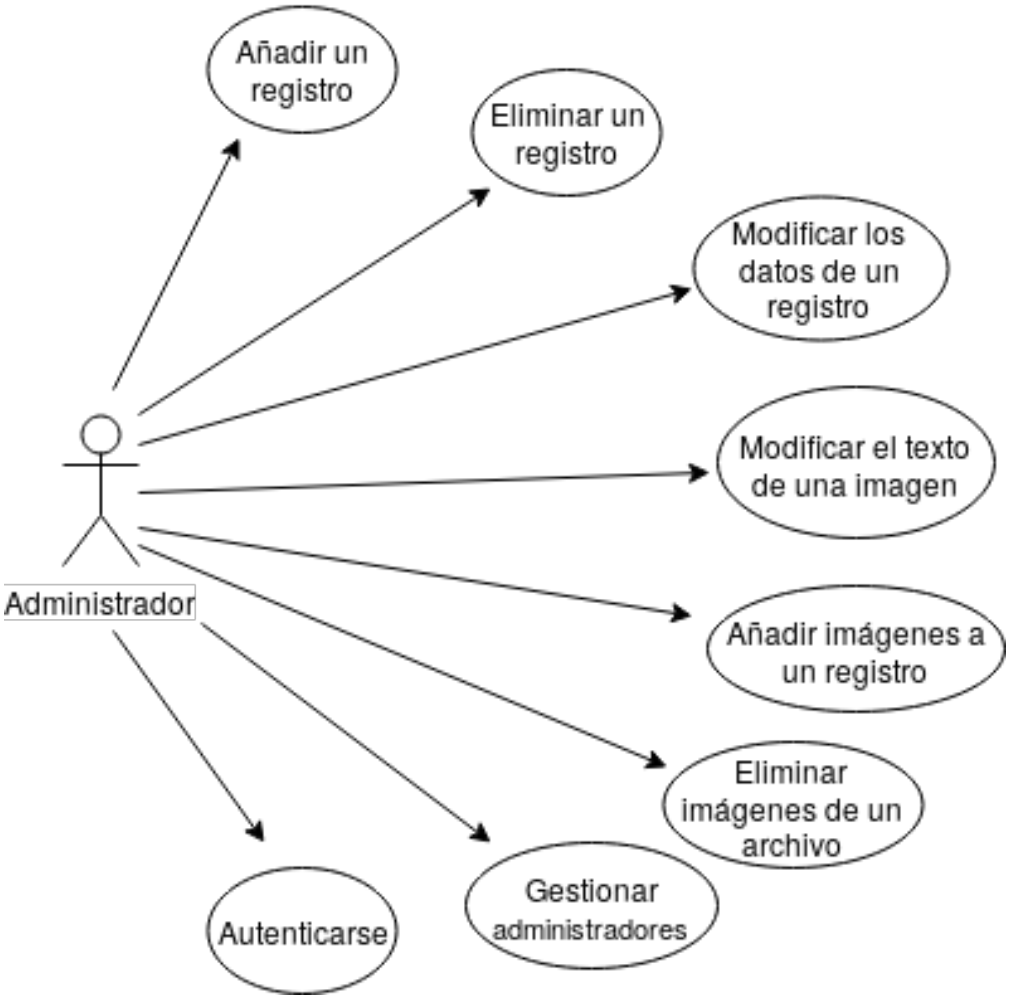

Figura 10: Diagrama de Casos de Uso para administrador

En ella se muestran dos actores: el administrador y el usuario. Tanto el usuario como el administrador pueden realizar búsquedas en el sistema, aunque para el usuario es la única posibilidad. Además, por diseño no se contempla la posibilidad de que un usuario se pueda registrar por cuenta propia en el sistema

El administrador, además, es el encargado de gestionar el sistema creando, modificando y eliminando registros, y usuarios. En el siguiente apartado, Descripción de los casos de uso (3.3), se describirán en detalle cada uno de estos roles y casos de uso asociados

3.3. Descripción de los Casos de uso

En este apartado se muestra una descripción formal de casos de uso. El formato que se seguirá para realizar esta descripcion se muestra en el Cuadro 2. Como se

puede ver, por cada caso de uso, se describirán el título, el actor que interviene en el caso de uso, las precondiciones para poder llevarlo a cabo, y una descripción del mismo.

ID	CU-XX
Título	
Actor	
Precondiciones	
Descripción	

Cuadro 2: Plantilla de casos de uso

Como se describía anteriormente, hay dos actores definidos en el sistema: Administrador y usuario. El administrador tiene todas las opciones de un usuario, y además tiene permisos para realizar operaciones CRUD (Create, Read, Update and Delte), tanto en registros, como en usuarios, mientras que el segundo. El usuario solo puede buscar texto en las imágenes de los registros. La diferencia entre ambos es que en el portal un administrador puede autenticarse, teniendo la posibilidad de acceder a la parte de administrador, y un usuario no necesita autenticarse para poder realizar búsquedas.

ID	CU-01
Título	**Añadir un registro**
Actor	Administrador
Precondiciones	Estar autentificado en el sistema
Descripción	El administrador suministra al sistema un registro, que consiste de Título, Autor, Año, y una o más imágenes

Cuadro 3: Caso de usuario CU-01

ID	CU-02
Título	**Eliminar un registro**
Actor	Administrador
Precondiciones	Estar autentificado, y el registro existe en la base de datos
Descripción	El administrador elimina un registro existente

Cuadro 4: Caso de usuario CU-02

ID	CU-03
Título	**Modificar los datos de un registro**
Actor	Administrador
Precondiciones	Estar autentificado, y el registro existe en la base de datos
Descripción	El administrador puede editar el título, al autor y/o el año de un registro.

Cuadro 5: Caso de usuario CU-03

ID	CU-04
Título	**Modificar el texto de una imagen**
Actor	Administrador
Precondiciones	Estar autentificado, y el registro existe en la base de datos
Descripción	El administrador podrá editar el texto procesado por el algoritmo de Machine Learning de las imagenes de un registro.

Cuadro 6: Caso de usuario CU-04

ID	CU-05
Título	**Añadir imágenes a un registro**
Actor	Administrador
Precondiciones	Estar autentificado, y el registro existe en la base de datos
Descripción	El administrador puede añadir imágenes a un registro que ya tuviera imágenes, y solo las añadidas en ese momento serán procesadas por el sistema para búsqueda de texto.

Cuadro 7: Caso de usuario CU-05

ID	CU-06
Título	**Eliminar imágenes de un registro**
Actor	Administrador
Precondiciones	Estar autentificado, y el registro existe en la base de datos
Descripción	El administrador puede borrar imagenes de un registro, pero tiene que haber una imagen mínimo por registro.

Cuadro 8: Caso de usuario CU-06

ID	CU-07
Título	**Buscar con texto**
Actor	Usuario, Administrador
Precondiciones	-
Descripción	El usuario puede introducir en el campo de búsqueda texto (una palabra, o frase), y se mostrarán los registros que contengan imágenes con ese texto buscado

Cuadro 9: Caso de usuario CU-07

ID	CU-08
Título	**Buscar con texto y parámetros de búsqueda**
Actor	Usuario, Administrador
Precondiciones	Registros existentes
Descripción	Siguiendo el caso de Usuario CU-07, añadiendo como parámetro de búsqueda el autor, y/o el año

Cuadro 10: Caso de usuario CU-08

ID	CU-09
Título	**Buscar con texto y/o parámetros de búsqueda, y/o ordena por parámetros**
Actor	Usuario, Administrador
Precondiciones	Registros existentes
Descripción	Siguiendo el caso de Usuario CU-08, añadiendo como parámetro de ordenación, título, autor, y/o el año

Cuadro 11: Caso de usuario CU-09

ID	CU-10
Título	**Gestionar administradores**
Actor	Administrador
Precondiciones	
Descripción	Un administrador puede añadir (con nombre de usuario, correo y contraseña), editar, buscar, o borrar, otro usuario administrador.

Cuadro 12: Caso de usuario CU-10

ID	CU-11
Título	**Autenticarse**
Actor	Administrador
Precondiciones	-
Descripción	Un administrador se logea en el sistema con credenciales nombre de usuario y contraseña

Cuadro 13: Caso de usuario CU-11

3.4. Requisitos

Una vez definidos los casos de uso, podemos diseñar los requisitos funcionales (RF), y no funcionales (RNF). Similarmente al caso anterior, el Cuadro 14 muestra la plantilla utilizada para presentar los requisitos.

ID	RF-XX || RNF-XX
Título	
Descripción	
Prioridad	
Caso(s) de uso	

Cuadro 14: Plantilla de requisitos

Por cada requisito se especifica un identificador, el título, una descripción, su prioridad, y los casos de uso con los que está relacionado. Sobre la prioridad se especifican tres categorías: alta, media o baja.

3.4.1. Requisitos funcionales

En este apartado, se presentan los requisitos funcionales del sistema desarrollado.

ID	RF-01
Título	**Búsqueda de texto en imágenes**
Descripcion	Mediante el uso de Google Vision API (que a su vez, usa NN) procesar una imágen en búsqueda de texto, con un previo preprocesado previo
Prioridad	Alta
Caso(s) de uso	CU-01, CU-04, CU-07, CU-08, CU-09

Cuadro 15: Requisito funcional RF-01

ID	RF-02
Título	**Base de datos**
Descripcion	Para el almacenamiento de los registros, se requiere una base de datos, siguiendo el esquema mostrado en la imagen 13
Prioridad	Alta
Caso(s) de uso	CU-01, CU-02, CU-03, CU-04, CU-05, CU-06, CU-07, CU-08 , CU-09, CU-10, CU-11

Cuadro 16: Requisito funcional RF-02

ID	RF-03
Título	**Portal**
Descripcion	Se debe diseñar un portal web que implemente la funcionalidad requerida
Prioridad	Alta
Caso(s) de uso	CU-01, CU-02, CU-03, CU-04, CU-05, CU-06, CU-07, CU-08 , CU-09, CU-10, CU-11

Cuadro 17: Requisito funcional RF-03

ID	RF-04
Título	**Búsqueda en texto**
Descripcion	En el portal (RF-03) un usuario puede buscar texto en las imágenes subidas por el administrador(RF-08), y procesadas (RF-01). Adicionalmente, puede filtrar la búsqueda por autor y año, y/o ordenar la búsqueda por título, autor y año
Prioridad	Alta
Caso(s) de uso	CU-07, CU-08, CU-09

Cuadro 18: Requisito funcional RF-04

ID	RF-05
Título	**Acceso a portal administrador**
Descripcion	Un usuario que tenga credenciales, puede logearse en el portal, para acceder a la parte de administrador
Prioridad	Alta
Caso(s) de uso	CU-01, CU-02, CU-03, CU-04, CU-05, CU-06, CU-10

Cuadro 19: Requisito funcional RF-05

ID	RF-06
Título	**Edición de registro**
Descripcion	En la sección de administrador (RF-05) se puede seleccionar un registro, para editar sus datos (Título, Autor, Año)
Prioridad	Media
Caso(s) de uso	CU-03

Cuadro 20: Requisito funcional RF-06

ID	RF-07
Título	**Edición de registro**
Descripcion	En la sección de administrador (RF-05) se puede seleccionar un registro, para editar el texto de las imágenes, y añadir o borrar una imágen
Prioridad	Media
Caso(s) de uso	CU-04, CU-05, CU-06

Cuadro 21: Requisito funcional RF-07

ID	RF-08
Título	**Añadir un registro**
Descripcion	En la sección de administrador (RF-05), se puede añadir un registro, necesitando como datos obligatorios el título, autor, y año, y al menos una imágen. El sistema procesará las imágenes añadidas, y guardará los datos en la base de datos.
Prioridad	Alta
Caso(s) de uso	CU-01

Cuadro 22: Requisito funcional RF-08

ID	RF-09
Título	**Eliminar un registro**
Descripcion	En la sección de administrador (RF-05), se puede eliminar un registro, con un diálogo de confirmación. El sistema borrará los datos de la base de datos.
Prioridad	Media
Caso(s) de uso	CU-02

Cuadro 23: Requisito funcional RF-09

ID	RF-10
Título	**Corregir texto de una imágen**
Descripcion	En la sección de administrador (RF-05), después de subir imágenes, se da la opción a corregir el texto procesado por el sistema, así como eliminar imágenes del registor (como mínimo el registro ha de tener una imagen)
Prioridad	Baja
Caso(s) de uso	CU-04, CU-06

Cuadro 24: Requisito funcional RF-10

ID	RF-11
Título	**Añadir un administrador**
Descripcion	En la sección de administrador (RF-05), un administrador puede añadir otro administrador, con nombre de usuario, email y contraseña
Prioridad	Media
Caso(s) de uso	CU-10

Cuadro 25: Requisito funcional RF-11

ID	RF-12
Título	**Gestionar administradores**
Descripcion	En la sección de administrador (RF-05), un administrador puede editar otro administrador, y buscar otros administradores. Si edita sus propios datos, tiene que hacer login de nuevo
Prioridad	Baja
Caso(s) de uso	CU-10, CU-11

Cuadro 26: Requisito funcional RF-12

ID	RF-13
Título	**Añadir un administrador**
Descripcion	En la sección de administrador (RF-05), un administrador puede eliminar otro usuario (tras previa confirmación)
Prioridad	Baja
Caso(s) de uso	CU-10

Cuadro 27: Requisito funcional RF-13

ID	RF-14
Título	**Login**
Descripcion	Un usuario administrador debe poder autenticarse en el sistema
Prioridad	Alta
Caso(s) de uso	CU-10

Cuadro 28: Requisito funcional RF-14

ID	RF-15
Título	**Resultados de búsqueda resaltados**
Descripcion	Tras realizar una búsqueda con texto, las imágenes que contengan el texto buscado serán resaltadas. Si se realiza una búsqueda sin texto, ninguna imagen será resaltada
Prioridad	Media
Caso(s) de uso	CU-07, CU-08, CU-09

Cuadro 29: Requisito funcional RF-15

ID	RF-16
Título	**Validación de año de registro**
Descripcion	El año de creación de registro se valora de la siguiente forma: No puede ser mayor que el año actual, no puede empezar por cero, a no ser que sea el año 0, y los años AC tienen que tener el formato -<*año*>
Prioridad	Media
Caso(s) de uso	CU-01, CU-03

Cuadro 30: Requisito funcional RF-16

ID	RF-17
Título	**Validación de contraseña**
Descripcion	En creación y edición de usuarios, la contraseña a crear o modificar se debe proveer dos veces, y se validará que la segunda contraseña sea correcta.
Prioridad	Media
Caso(s) de uso	CU-10

Cuadro 31: Requisito funcional RF-17

ID	RF-18
Título	**Tras realizar la búsqueda, la lista de resultados es presentada**
Descripcion	Tras realizar una búsqueda, con o sin texto, con o sin parámetros de búsqueda, con o sin parámetros de ordenación, se presenta una lista de registros que contienen cada uno el título del registro, el autor, el año de creación, y las imágenes que lo componen
Prioridad	Alta
Caso(s) de uso	CU-07, CU-08, CU-09

Cuadro 32: Requisito funcional RF-18

ID	RF-19
Título	**Vista detallada de imágenes**
Descripcion	En la lista de resultados que se presenta tras hacer una búsqueda (RF-18), tras pinchar en cualquier lado del registro (si no es una imagen), se muestran las imágenes desde la primera. Si se pincha una imagen, se muestran las imágenes desde la que se ha pinchado
Prioridad	Alta
Caso(s) de uso	CU-07, CU-08, CU-09

Cuadro 33: Requisito funcional RF-19

ID	RF-20
Título	**Búsqueda de administradores**
Descripcion	En la sección administración del portal, se permite a un administrador hacer búsquedas sobre el nombre de usuario y sobre el email. Se permite ordenar por fecha de creación, nombre de usuario, e email.
Prioridad	Baja
Caso(s) de uso	

Cuadro 34: Requisito funcional RF-20

3.4.2. Requisitos no funcionales

ID	RNF-01
Título	**Python**
Descripcion	Tener Python 3.6+ instalado
Prioridad	Alto

Cuadro 35: Requisito no funcional RNF-01

ID	RNF-02
Título	**Sistema Operativo**
Descripcion	El sistema es capaz de ser ejecutado en cualquier sistema con arquitectura UNIX (Distribuciones de Linux, y de Mac)
Prioridad	Alto

Cuadro 36: Requisito no funcional RNF-02

ID	RNF-03
Título	**Módulos para la parte servidor (Backend)**
Descripcion	Tener Django 2.2, Django-Rest-Framework 3.9.2+, Google Cloud Vision 0.36 instalados
Prioridad	Alto

Cuadro 37: Requisito no funcional RNF-03

ID	RNF-04
Título	**Módulos para la parte cliente (Frontend)**
Descripcion	Tener reactfront 16.8+, axios 0.18+, react-dropzone 10.1+, rc-slider 8-6+, react-image-lightbox 5.1+ instalados
Prioridad	Alto

Cuadro 38: Requisito no funcional RNF-04

ID	RNF-05
Título	**Imágenes nítidas**
Descripcion	Las imágenes subidas por el administrador deben ser lo más nítidas posible. imágenes con mucho ruido de fondo pueden dar problemas. De no ser así, la traducción de la imagen a texto puede no realizarse de forma correcta [Hosseini et al., 2017].
Prioridad	Medio

Cuadro 39: Requisito no funcional RNF-05

ID	RNF-06
Título	**Tamaño de la Base de Datos**
Descripcion	La capacidad de registros a almacenar depende del hardware en el que esté hosteado el servidor de la base de datos
Prioridad	Media

Cuadro 40: Requisito no funcional RNF-06

3.4.3. Matriz de trazabilidad de casos de uso

Para entender mejor la relación entre casos de uso, y requisitos, a continuación se muestra una matriz de trazabilidad entre ambos. En esta matriz podemos ver qué requisitos satisfacen qué casos uso. No se incluyen requisitos no funcionales, ya que estos son precondiciones para que el sistema funcione.

	UC-01	UC-02	UC-03	UC-04	UC-05	UC-06	UC-07	UC-08	UC-09	UC-10	UC-11
RF-01	✓			✓				✓	✓		
RF-02	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
RF-03	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
RF-04							✓	✓	✓		
RF-05	✓	✓	✓	✓	✓	✓				✓	
RF-06			✓								
RF-07				✓	✓	✓					
RF-08	✓										
RF-09		✓									
RF-10				✓	✓						
RF-11										✓	
RF-12										✓	✓
RF-13										✓	
RF-14											✓
RF-15							✓	✓	✓		
RF-16										✓	
RF-17										✓	
RF-18							✓	✓	✓		
RF-19							✓	✓	✓		

Cuadro 41: Matriz de trazabilidad de casos de uso

4. Diseño del sistema

En los siguientes apartados se se describe en primer lugar de forma global los principales componentes de la arquitectura (Sección 4.1) desarrollada y cómo interaccionan entre ellos. Posteriormente se describen en detalle cada uno de los componentes,con la descripción de la parte Back (Sección 4.2), que incluye, además, la descripción de la base de datos (Sección 4.2.2). También se describe el componente Front (Sección 4.3) y, por último, la descripción del componente de Machine Learning encargado de hacer la traducción de las imágenes a texto (Sección 4.4.).

4.1. Arquitectura del sistema

Como se ha mencionado previamente, hay cuatro componentes principales del sistema, Base de Datos (BBDD) PostgreSQL hosteada en un servidor externo, un backend Python-Django, un servidor frontend Node-React, y el proceso de detección de texto mediante Machine Learning, usando Google API. A continuación se muestra cómo interaccionan los diferentes componentes

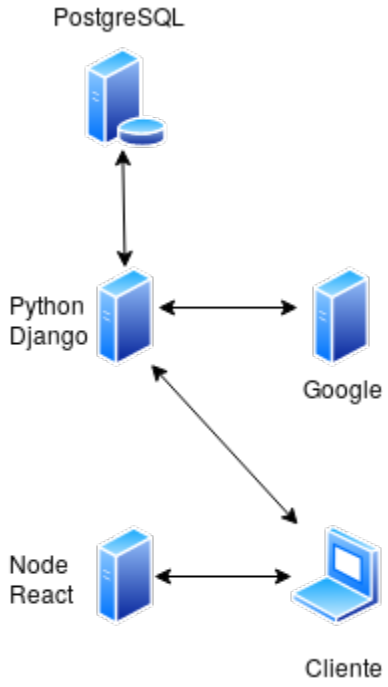

Figura 11: Arquitectura del sistema

Como se ha mencionado previamente, este sistema se caracteriza por ser desacoplado, lo cual incrementa la modularidad y la seguridad. El servidor de front (Node React) no tiene conexión con nada más que el cliente, y éste a su vez

solo se conecta con el servidor back. El ”grueso” de seguridad se maneja desde la API, y mediante prácticas llevadas a cabo (que se detallan en la siguiente sección), por lo que se reducen las vulnerabilidades, y se incrementa la flexibilidad a la hora de desarrollar.

4.2. Back - Django-Rest-Framework

Siguiendo la estructura de Django, el proyecto tiene el siguiente elenco de carpetas y archivos:

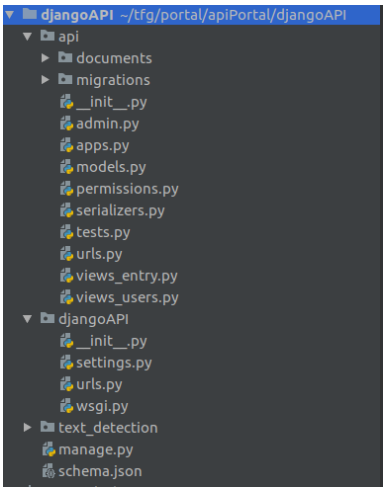

Figura 12: Sistema de ficheros en Django

Debido al enfoque desacoplado, en vez de usar Django ”tal y como viene”, se usará para el proyecto su versión API REST, Django-Rest-Framework (DRF). Como se ha mencionado antes, Django usa un sistema modular de aplicaciones, DRF es una aplicación que se puede instalar en un proyecto de Django, y permite convertirlo en una API.

Una API REST (Representational State Transfer) es un tipo de arquitectura de software que provee un estándar entre sistemas en la red, facilitando la comunicación entre los mismos. Sistemas REST se caracterizan por no mantener un estado, y separar el cliente del servidor. Este tipo de sistemas funcionan recibiendo peticiones, que consisten de:

- Un *verbo* HTTP, que define qué tipo de operación. Estos verbos sirven para realizar operaciones CRUD (Create Read Update Delete) en la base de datos. Los 4 más comunes son
    - GET, para obtener un recurso específico (por id) o una colección de recursos

- POST, para crear un recurso nuevo
- PUT, actualiza un recurso específico
- DELETE, borra un recurso específico

- Una cabezera, que contiene información variada, como por ejemplo, el tipo de contenido que es aceptada por el servidor.
- Un camino al recurso, o comunmente llamado *endpoint*
- Opcionalmente, un cuerpo del mensaje que contiene datos a procesar

4.2.1. "Flujo" de Django

Una vez visto la tecnología que va a ser utilizada, y metología (DRF, y REST respectivamente), cabe hacer una pequeña introducción a la forma en la que funciona Django, desde la creación de un modelo a la base de datos, hasta que es enviado al recibir una petición. En total hay cuatro "pasos":

1. **Modelo**
   Es la unidad básica que representa una Tabla (o un objeto) en la base de datos. Naturalmente, en estos modelos se pueden guardar datos de formas diversas, campo de texto, campo de fecha, enteros..., incluyendo relaciones, y claves extranjeras.
   Por ejemplo, si quisieramos tener una tabla que guarde Personas, con nombre y apellido, se haría lo siguiente:

Código 3: Modelo de Person en Django

```
from django.db import models

class Person(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
```

Crea una tabla de datos de esta forma:

Código 4: Tabla de Person en SQL

```
CREATE TABLE myapp_person (
    "id" serial NOT NULL PRIMARY KEY,
    "first_name" varchar(30) NOT NULL,
    "last_name" varchar(30) NOT NULL
);
```

Aquí es donde se pueden empezar a ver las ventajas de Django, en el ejemplo de código 3, están implícitas todas las operaciones CRUD, Create (save()), Read (equivalente a get()), Update (udpate()), y Delete (delete()), además de todas los métodos típicos de un objeto en python, por ejmplo, el valor String (__str__()). Si se necesita realizar cualquiera de las operaciones de forma diferente a lo esperado, es tan sencillo como hacer un overriding del método. Se dará un ejemplo de esto en la subsección Modelos del Proyecto, dentro de esta sección.

2. **Serializer**
   En aquitecturas REST, es muy común trabajar con estándares de representación de datos, por ejemplo, JSON. Los modelos de Django son estructuras de datos complejos, no estandarizados, por lo que, a la hora de tratarlos en forma de JSON, por ejemplo, hay que "traducirlos" desde su compleja estructura a algo más sencillo, y aquí es donde entra un serializer.
   Los serializers de este proyecto vienen de DRF, y no de Django, y sirven además de traductores, de validadores de datos. Si un campo en concreto de un modelo debe recibir una fecha, y recibe un número, el validador dará como resultado inválido, y no permitirá guardar. Se verán ejemplos de serializers usados en la subsección Serializers.

3. **View**
   Las views en Django, como se ha mencionado previamente, son las unidades de control que reciben el input del usuario desde el cliente, y ejecutan lógica dependiendo de ello. En DRF existen function-based views, métodos atómicos que sirven un propósito concreto, con la ayuda de decoradores (la versión Pythónica del patrón de diseño Proxy), se pueden extender para permitir diferentes acciones, permisos... Y existen class-based views, que es una forma relativamente más sencilla que las function-based, en la que viene todo hecho, por lo que, como pasaba con los modelos, a no ser que se quiera algo específico, en muy pocas líneas se puede tener un controlador que realice todas las operaciones CRUD.
   Las vistas que usen cualquier modelo de la base de datos, tiene que usar el Serializer correspondiente primero para traducir los datos a String/JSON... Como en los casos anteriores, se mostrarán ejemplos en la subsección Endpoints.

4. **Routing**

Por último, la parte de routing. Con el sistema de Proyecto-aplicaciones que tiene Django, la forma de dirigir las peticiones a la aplicación adecuada, en cada aplicación se definen las URLs (endpoints) de esa aplicación, y por cada URL, se especifica qué view es la que tiene que ser llamada. En la especificación de URLs del proyecto se puede importar cada una de las URLs de las aplicaciones, haciendolo así modular y limpio. En la sección *endpoints* se hablará de los endpoints creados para el proyecto.

Como se puede observar, hay un flujo subyacente bidireccional en la forma de trabajar con Django, primero se crean los *modelos*, después el *serializer* del modelo, luego la *view* que usará ese serializer, y por último el *endpoint* con la view asociada. Cuando Django recibe una petición, consulta los endpoints, el endpoint tiene la view, en la view se haya la lógica, junto con el serializer, y el serializer usa el modelo para validar los datos.

4.2.2. Modelos

Para realizar los modelos, se ha seguido el modelo de la base de datos mostrado en la imagen 13

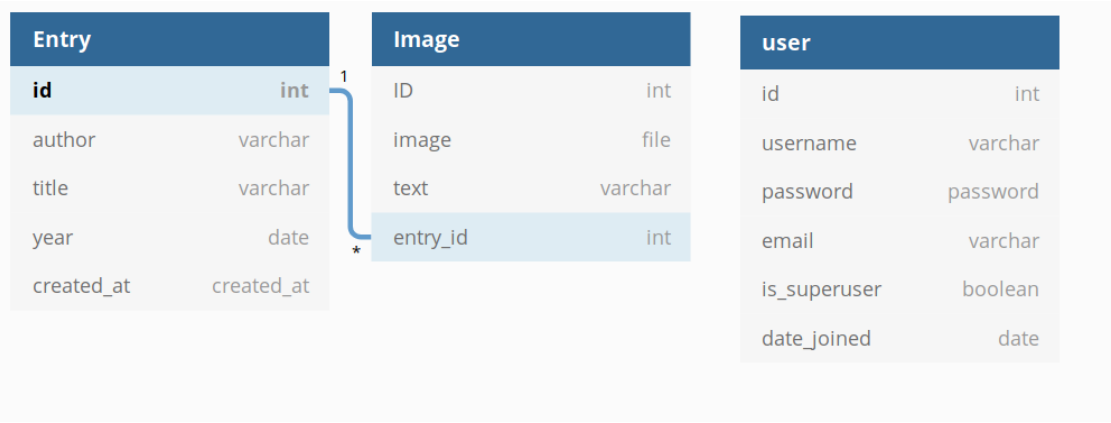

Figura 13: Modelo de base de datos

En el archivo *models.py* nos encontramos entonces con dos modelos, *Entry*, que corresponde a un registro, e *Image*, que corresponde con imágenes. Para los usuarios, en el framework Django viene implementado un sistema de usuarios.

User Para el usuario, el modelo de Django viene con más campos de los necesarios para este proyecto, ya que incluye campos last_login, first_name, last_name, is_active, e is_staff, que no son usados, pero incluye campos id, password, is_superuser, username, email, date_joined que sí son usados. Adicionalmente,

el sistema de usuarios incluye almacenamiento de contraseñas guardadas en formato `<algorithm>$<iterations>$<salt>$<hash>`
El algoritmo usado es PBKDF2 (Password-Based Key Derivation Function), aplica una función pseudorandom, dada una contraseña, con un valor *salt*, y repite el proceso muchas veces hasta producir una clave derivada. Inicialmente el valor de repeticiones recomendado era de 1000[Kaliski, 2000], actualmente se pueden hacer alrededor de 100.000 iteraciones en un servidor sin gran coste computacional. De hecho, Django con este algoritmo ejecuta 150.000 repeticiones.

Dado lo seguro que es usar el modelo de usuario por defecto de Django, se ha optado por no implementar uno propio.

Entry Un registro está definido por título, autor, año, y fecha de creación (del registro en la base de datos). Tiene además una relación 1-N con imágenes, por cada registro puede haber varias imágenes (en la especificación de requisitos está recogido que debe haber por lo menos una imagen por registro). La forma de especificar estos campos en Django es:

Código 5: Modelo Entry en Django

```
class Entry(models.Model):
    title = models.CharField(max_length=256)
    author = models.CharField(max_length=128)
    year = models.fields.IntegerField()
    created_at = models.DateTimeField(auto_now_add=True)
```

Se pueden observar varias cosas, la primera, no se crea la relación 1 a n en el modelo Entry en sí, se crea en el modelo Image. Se establece una longitud máxima de autor y título de 128 y 256 caracteres respectivamente, y el campo de fecha de creación del registro en la base de datos se popula automáticamente, no viene dado por un tiempo arbitrario dado desde el cliente.

En SQL este modelo equivale a

Código 6: Tabla Entry en SQL

```
CREATE TABLE "api_entry" (
  "id" serial NOT NULL PRIMARY KEY,
  "title" varchar(256) NOT NULL,
  "author" varchar(128) NOT NULL,
  "year" integer NOT NULL,
  "created_at" timestamp with time zone NOT NULL
```

```
  );
```

Image Por último, el modelo Image. Image tiene como campos la imagen en sí, el texto procesado de la imagen, y la entrada a la que pertenece. En Django queda de la siguiente forma:

Código 7: Modelo Image en Django

```
class Image(models.Model):
    entry = models.ForeignKey(Entry, on_delete=models.CASCADE, related_name='
        entry_images')
    image = models.ImageField(upload_to='documents/%Y/%m/%d')
    text = models.TextField(default="Texto no encontrado en la imagen")

    def save(self, *args, **kwargs):
        if "override" not in kwargs:
            self.text = settings.DETECTOR(self.image.read())
        else:
            del kwargs["override"]

        return super(Image, self).save(*args, **kwargs)
```

En este caso, a parte de los campos pertinentes (incluyendo la foreign key de imagen a entry), hace falta sobreescribir el método save del modelo para procesar la imagen. Cuando se llama a la función save (a veces de forma implícita, desde el objeto en sí, a veces desde un Serialiazer), si no contiene el parámetro override, se llamará a la función DETECTOR, que es la implementación de Machine Learning utilizado para este proyecto.

Se añade además la posibilidad de recibir "override", con lo cual se guardará la imagen tal y como viene. Esto es necesario para el caso en el que un usuario administrador esté editando el texto de una imagen.

Este modelo en Django equivale en SQL a:

Código 8: Tabla Image en SQL

```
CREATE TABLE "api_image" (
  "id" serial NOT NULL PRIMARY KEY,
  "entry_id" integer NOT NULL,
  "image" varchar(100) NOT NULL,
```

```
  "text" text NOT NULL
);

CREATE INDEX "api_image_entry_id_7287b5c0" ON "api_image" ("entry_id");

ADD CONSTRAINT "api_image_entry_id_7287b5c0_fk_api_entry_id" FOREIGN KEY ("
    entry_id") REFERENCES "api_entry" ("id") DEFERRABLE INITIALLY DEFERRED;
```

La imagen en sí, como se puede observar en el campo imagen (varchar (100)), no es guardada en la base de datos, esto es porque por lo general se considera que las imágenes ocupan mucho, por lo que pueden ralentizar la base de datos rápidamente. Idealmente, en un entorno profesional, el proceso más óptimo sería guardar las imágenes en un servicio externo, y en la base de datos guardar la referencia a ese servicio externo. Para este proyecto, ya que no se prevee una gran cantidad de imágenes durante la fase de desarrollo y testeo, se ha configurado de forma que la imagen se guarda en el propio servidor web.

4.2.3. Serializers

Como se ha mencionado previamente, los Serializers son los encargados de "traducir"de un formato complejo a uno más sencillo (típicamente JSON/XML/String). Sirven también para validación de datos, y obtener un valores de forma deseada. Los serializers usados en este proyecto son:

1. **ImageSerializer**

Código 9: ImageSerializer

```
class ImageSerializer(serializers.ModelSerializer):
  class Meta:
      model = Image
      fields = ('id', 'image', 'text', 'entry')
```

Como se puede observar, de nuevo Django provee facilidades a la hora de desarrollar. Con estas 4 líneas de código, tenemos toda la funcionalidad descrita respecto a serializers, valida que un objeto tenga los campos *fields*, y se encarga de llamar a las distintas funciones del modelo Image (create, save, update...) Se ha usado también una versión "lite" de este serializer, en el que la entry no está incluída como field, para el el serializer EntrySerializer, ya que en caso de dejar el campo entry en el ImageSerializer, se incurriría en una referencia circular.

2. **EntrySerializer**

Código 10: EntrySerializer

```
class EntrySerializer(serializers.ModelSerializer):
  entry_images = ImageSerializerLite(many=True, read_only=True)

  class Meta:
    model = Entry
    fields = ('id', 'title', 'author', 'year', 'created_at', 'entry_images
        ')
```

En este caso, la invocación de entry_images como ImageSerializer significa que, cuando el EntrySerializer sea llamado, va a hacer un lookup en la tabla Image, y devolverá todas las imágenes que correspondan a cada registro.

3. **UserSerializer** Dentro de esta categoría existen dos serializers distintos, el general (código 11), usado para devolver valores desde la base de datos, en el que expresamente se omite la contraseña, por lo que no se envían nunca contraseñas desde el servidor al cliente, y un serializer detallado (código 12), usado para validar cuando un usuario es creado o editado, en el que se además se crea la contraseña con el algoritmo seguro de Django.

Código 11: UserSerializer

```
class UserSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = ('username', 'email', 'id', 'date_joined')
```

Código 12: UserDetailedSerializer

```
class UserDetailedSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = ('username', 'email', 'id', 'date_joined', 'password')

    def validate_password(self, value):
        # Password created using Django secure method
        return make_password(value)

    def create(self, validated_data):
        user = User.objects.create_superuser(
```

```
            username=validated_data['username'],
            email=validated_data['email'],
            password=validated_data['password'],
        )

        user.save()

        return user
```

4.2.4. Views

Como se ha mencionado antes, en Django las Views no equivalen a las vistas en el típico modelo MVC, se asemejan más al controlador. En este proyecto, debido a que es una API REST, en vez de seguir le modelo templating de Django, la mayoría del peso de implementación y desarrollo cae en esta parte, las views.

Hay dos partes de la API claramente diferenciadas, todo lo relacionado con los usuarios, y todo lo relacionado con las entradas, es por esto que se ha decidido separar la lógica de cada parte en dos ficheros distintos, *views_entry.py*, y *views_users.py*. Como mención especial, todas las vistas que se detallan a continuación (salvo que se diga lo contrario), usan un sistema de comprobación de autentificación, mediante el cual se comprueba que en la cabecera de la petición exista un token válido. De no existir, se devuelve un error HTTP 401 Unauthorized (no autorizado) sin realizar ninguna otra operación.

Un aspecto que cabe mencionar sobre Django es que las queries en Django a base de datos crean un objeto llamado queryset. Una queryset se encarga de ejecutar queries contra la base de datos, y se caracteriza porque es *lazy*, lo cual quiere decir que si se programa una query a través de este mecanismo, hasta que no se necesite el resultado no se va a ejecutar. Además, una queryset devuelve siempre otra queryset, esto permite realizar múltiples operaciones de filtrado y ordenación de forma no costosa, ya que no se ejecutará hasta que sea estrictamente necesario. Adicionalmente, este mecanismo ayuda a proteger contra SQLinjection, ya que se está tratando con un objeto, en vez de realizar la query con un String, y la parte de creación de la query, y la "traducción" de dicha query a sentencias SQL están separadas.

Entry views

Las cuatro views de registros son:

- Entry View: la vista principal del proyecto. Esta vista permite dos tipos

de peticiones, *GET*, y *POST*, ya que atiende a las peticiones de búsqueda, y de creación de registros. Esta vista tiene un modelo de autentificar personalizado, creado en un fichero separado (*permissions.py*), en el cual se comprueba que la petición es de tipo *GET*, o si el usuario realizando la petición está autentificado. Si ambas opciones son falsas (no es de tipo *GET*, ni el usuario está autentificado), la autentificación falla y la vista devuelve un error HTTP 401 Unauthorized (no autorizado). Adicionalmente, esta class-based view tiene dos parsers activos, JSONParser, el que viene por defecto que parsea una petición con contenido JSON, y MultiPartParser, se utiliza para parsear contenido *form-data* de la petición. Este form-data puede contener pares clave-valor, incluyendo ficheros mandados en el cuerpo de la petición.

La petición de tipo *GET* corresponde con las peticiones de búsqueda, incluyendo texto, parámetros de búsqueda (filtros), y parámetros de ordenación. En esta vista se recogen los diferentes parámetros de búsqueda, texto, filtros y ordenación. En este caso la búsqueda se hace de forma diferente a simplemente obtener una lista de registros, debido a que se necesita poder buscar una palabra o una frase completa. Se ha tomado la decisión por diseño de realizar la búsqueda de la siguiente forma:

Código 13: Query del texto recibido

```
entries = Entry.objects.all()
if q:
    _q = q.split(" ")
    _q.append(q)
    query = functools.reduce(operator.or_, (Q(
        entry_images__text__unaccent__icontains=item) for item in _q))
    entries = entries.filter(query)

    # Get the img ids that correspond to the text found
    query = functools.reduce(operator.or_, (Q(text__unaccent__icontains=
        item) for item in _q))
    image_ids = Image.objects.filter(query).values_list('id', flat=True)
```

El parámetro q es el texto que el usuario ha pedido buscar, y este puede ser una palabra o varias. Se ha considerado adecuado buscar tanto por palabras individuales como por la frase entera, por lo que $\_q$ tendría ese contenido. Por ejemplo, si un usuario busca "librero del rey", la petición llegará com parámetro *q="librero del rey"*, tras lo cual $\_q$ almacenará ['librero', 'del', 'rey', 'librero del rey'].

Django permite hacer las queries de varias formas, una de ellas es mediante las Q queries, que permite realizar agrupar varias queries en una. Haciendo uso de esta funcionalidad, para hacer una búsqueda sobre todos estos términos, se usa *reduce*, junto con el operador OR, para juntar todos los items de _ *q*, en una búsqueda que **ignora los acentos y busca si el texto contiene** lo buscado, lo que permite buscar palabras incompletas ("librer" dará el mismo resultado que "librero"). Adicionalmente, para el resaltado de imágenes, se devuelve también en esta vista los ids de las imágenes que contienen el texto.
Tras este primer filtro sobre el texto, se realizan filtros adicionales por el filtro recibido en la petición de la siguiente forma:

Código 14: Queries de filtros adicionales

```
author = request.GET.get('author', '')
if author:
    entries = entries.filter(author__exact=author)
```

El código 14, que existe de la misma forma para el filtro por años, muestra la forma de comprobar si se ha pedido hacer un filtro sobre autor, y cuál es. En caso de no haber, el filtro no se ejecuta (ya que el condicional if devuelve falso), pero si la petición contiene un filtro, sobre la queryset hecho previamente con la búsqueda de texto, se filtra adicionalmente por autor (y, similarmente, por año en caso necesario). De nuevo, esta operación no contiene coste adicional al ser las queryset *lazy*.
De forma similar, para la forma de ordenar,

Código 15: Ordenación en queries

```
order_by_author = request.GET.get('order_author', '')
if order_by_author:
    _order_author = "author" if order_by_author.upper() == "ASCENDING"
        else "-author"
    _order_by.append(_order_author)
```

Por último, la petición de tipo *POST* corresponde con las peticiones de creación de un registro. Estas peticiones *tienen* que tener título, autor, año y por lo menos una imagen. Tras comprobar que los valores son válidos (mediante el uso de EntrySerializer), se crea el registro, y después cada imagen con la referencia al registro creado.

- Entry Detail: Es la vista que se encarga devolver, de actualizar los datos de un registro, título, autor y año (para actualizar imágenes, se utiliza la siguiente view, Image Update), o de borrar un registro. Permite por lo tanto peticiones de tipo *GET, PUT, PATCH* y *DELETE*
  Con *GET* simplemente obtiene el objeto de la base de datos y lo devuelve en una respuesta JSON
  Para *PUT*, y *PATCH* (actualización parcial, parecido a *PUT*), se obtiene la información de la petición, incluyendo el ID del registro a modificar. Se comprueba que tanto el ID como los valores sean válidos (mediante el Serializer), y se guarda el objeto con los cambios.
  Por último, *DELETE*, simplemente borra el registro.

- Image Update: esta vista sirve para modificar las imágenes de un registro en concreto. Recibe tanto *POST*, para añadir imágenes a un registro, como *PUT*, para actualizarlas. Si se recibe una petición que no es una de estas dos, se devuelve un estado HTTP 403, Forbidden (prohibido)
  En el *POST* se comprueba primero que el registro al que se refiere la petición existe. Si existe, se crea cada imagen individual (con lo que se activa el algoritmo de ML en el guardado del objeto), y por último, se extrae la información de estas nuevas imágenes creadas (ID, dirección de la imagen, nombre, y texto que ha sido reconocido automáticamente), y se devuelve esta información.
  Si la petición es de tipo *PUT*, la petición contiene imágenes para borrar, e imágenes cuyo texto hay que actualizar. Para esto se extrae del cuerpo de la petición las imágenes por un lado (que corresponden con las imágenes a actualizar), y por otro, *to_delete*, que contiene los IDS de las imágenes a borrar. Después, por cada una de estas dos listas, se itera realizando la acción pertinente: actualizar el objeto Imagen con el nuevo texto, activando el mecanismo *override* a la hora de guardar el objeto, para que el mecanismo de ML no actúe, y borrar el objeto Imagen.

- Entry Meta: por último, la view más sencilla de las cuatro. Esta vista, que sólo acepta *GET*, se encarga de extraer de la base de datos los valores de autores, año mínimo de creación de un registro, y año máximo de creación de un registro, y se devuelven en una respuesta JSON.

User views

Dentro de user views, hay tres vistas:

- UserView: haciendo uso de *viewsets* gracias a DRF, este tipo de views no

usan los típicos métodos (internos) get y post, usan métodos que siguen las siglas CRUD; tiene métodos create, retrieve, list (que actúa similar a retrieve), update, partial_update, y delete. La ventaja que proporciona usar esta clase es que simplemente hay que especificar si existe algún tipo de permiso que se tenga que comprobar, si se necesita usar algún parser (usará JSON parser por defecto), un valor *queryset*, que indica de dónde obtener los datos, y por último una serializer class. Por lo que, con el código 16, se obtiene todo el set de operaciones CRUD para el modelo User.

Código 16: UserView

```
class UserView(viewsets.ModelViewSet):
    permission_classes = (IsAuthenticated, )
    parser_classes = (JSONParser,)
    queryset = User.objects.all()
    serializer_class = UserDetailedSerializer
```

- UsersView: esta view está creada específicamente para servir las queries de búsqueda de usuarios. Estas queries son algo complejas, ya que tienen parámetros de búsqueda, paginación, y ordenación. Esta view sólo permite peticiones *GET*
  En esta view se obtienen primero el valor de todos los parámetros, o se establecen valores por defecto (por ejemplo, si no se envía el tamaño de paginación, el valor por defecto es 10). Después, se hace una búsqueda usando la funcionalidad FTS de Django y PostgreSQL, sobre los campos de usuario y email. Después, se resuelven todos los parámetros de ordenación, gracias a que Django permite ordenar por varios campos a la vez, y se devuelve tanto los usuarios como un valor de usuarios totales.

- UserDetailedView: por último, a propósito de mantener la sesión de un usuario, esta view, que sólo permite recibir peticiones *GET*, recibe un token, y si el token es válido, y pertenece a un usuario, se devuelven los datos (usuario y email) del usuario al que pertenece dicho token. Si el token no es válido, o no existe un usuario para dicho token, se devuelve un estado de error HTTP.

Hace falta mencionar que la view para autentificar a un usuario ya está implementada por defecto en Django, por eso no se ha mencionado como view en este apartado.

4.2.5. Endpoints

Los endpoints se separan en tres categorías, endpoints de registros, endpoints de usuarios, y endpoints de autentificación.

Registros

1. URL: **entries**/
   View: **EntryView**
   Razón: Este uno de los endpoints más imporantes del proyecto, se usa con dos propósitos, el primero, responder a queries de registros por parte del cliente, y el segundo tener un endpoint en el que enviar toda la información de un registro, título, autor, año, e imágenes, y procesarlo todo a la vez, sin tener que hacer varias peticiones, una por cada imagen, y luego una con los valores. De esta forma se simplifica tanto las llamadas realizadas, como la forma de desarrollar.

2. URL: **entry**/<**int:pk**>/
   View: **entry_detail**
   Razón: Tener un endpoint en el que poder realizar Borrado y Editado de datos, en el caso de edición de datos, solo sirve para editar los datos del registro en sí, pero no de las imágenes. Esto se ha hecho por diseño, ya que las funcionalidades de edición de ambas partes están separadas en el cliente.

3. URL: **entry_meta**/
   View: **entry_meta**
   Razón: A fin de crear los selectores de valor con los que poder filtrar en el cliente, es necesario tener los valores. Por ejemplo, de años, se devuelven el mínimo y el máximo, para hacer un year-picker.

4. URL: **images**/
   View: **image_update**
   Razón: Como se menciona en el endpoint de registros 2, entry/<int:pk>, la funcionalidad de edición de imágenes y la de edición de los datos del registro están separadas en el cliente, es por eso que tiene sentido que estén separadas también en el servidor. Este endpoint sirve para borrar, o editar, si se recive con PUT, o para añadir, si se recibe un POST.

Users

1. URL: **users**/
   View: **UsersView**
   Razón: Al igual que pasa con los registros, en la parte de administración de la interfaz, existe la posibilidad de buscar usuarios, filtrarlos, ordenarlos... Este endpoint sirve exclusivamente para eso.

2. URL: **user**/
   View: **UserView**
   Razón: Necesario para las operaciones create, edit, y delete de un usuario. Permite la actualización total o parcial (PUT, o PATCH)

Autentificación

1. URL: **login**/
   View: **obtain_auth_token**
   Razón: Este endpoint sirve para autentificar a un usuario tras recibir credenciales. Buscará o creará un token para el usuario si las credenciales son correctas.

2. URL: **user_meta**/
   View: **get_user_by_token**
   Razón: Tras recibir un token válido de un usuario, se devuelve la información del usuario al cliente. La intención de este endpoint es mantener la sesión de un usuario logueado cuando vuelve a meterse en el portal. **No** envía el campo de contraseña

4.2.6. Seguridad

A la hora de programar un servidor, hay que tener varias medidas de seguridad en cuenta, que pueden variar por diferentes aspectos, por ejemplo, un cliente acoplado de un desacoplado debe manejar diferente seguridad CRFS. A continuación se explican las diferentes medidas de seguridad que se han tomado a lo largo del desarrollo

Cross-Origin Resource Sharing - CORS Es un mecanismo que usa cabeceras HTTP adicionales para informar a un navegador que debe dejar a una aplicación corriendo en un origen, o dominio, tener permiso para acceder determinados recursos desde un servidor en un origen distinto. Una aplicación ejecuta una petición HTTP cross-origin cuando pide un recurso que tiene un diferente origen (dominio, protocolo, y puerto) que el su propio origen. [Mozilla, 2019]

Por ejemplo, un cliente de una aplicación con dominio *http://dominio-a.com*, realiza una petición a *http://dominio-b.com/data.json*. El dominio es distinto, por lo tanto, es una aplicación CORS.

Las peticiones, dentro de este protocolo de seguridad, se dividen en ”simples”, las que no activan el protocolo, que son GET, HEAD, y OPTIONS; y las peticiones denominadas *prefilghted*, que son aquellas que primero mandan una petición HTTP: OPTIONS al recurso del dominio extranjero, en función de la respuesta del servidor, se determina si se puede realizar la petición o no. Las peticiones que activan el procotolo (es decir, las *prefilghted*), son: PUT, DELETE; CONNECT, OPTIONS, TRACE, PATCH.

Para activar este protocolo de control en Django hay que realizar dos acciones, la primera, instalar *corsheaders*, que hace las veces de middleware, y se añade como aplicación, por lo que en el archivo de configuración de Django (settings.py en la carpeta de proyecto) hay que añadir ambos parámetros en las secciones correspondientes. La segunda medida a tomar es crear la lista de origenes permitidos. Ya que solo queremos permitir al servidor de front hacer peticiones, se añade el dominio del servidor a la whitelist. Con esta medida, se han restringido todas las peticiones que no vengan del dominio del cliente.

Cross-site Request Forgery - CSRF Un ejemplo de ataque CSRF,

Código 17: CSRF attack

```
<img src="https://bank.example.com/withdraw?account=bob&amount=1000000&for=
    mallory">
```

Como se puede observar, el atributo source de la imagen no es una imagen, es una petición de transferencia. Si la página sobre la que se ejecuta este ataque es un banco, y el usuario ya está autorizado (y, si muchas otras medidas de seguridad no estuviesen implementadas), el ataque tendría éxito, si no se protege contra CSRF.

CSRF funciona usando las cookies de sesión de un cliente, a la par que introduciendo código malicioso (al igual que CORS) en el cliente. Frente a lo segundo, al igual que con CORS, al tener un sistema ”cerrado”, se puede decir que es muy difícil que la vulnerabilidad venga por ese lado. Respecto a la primera, una forma implícita de proteger contra este tipo de ataques es usar token-based authentication, en vez de sesión, ya que el token se añade a la cabecera, mientras

que una sesión se guarda en una cookie, que siempre son enviadas automáticamente con una petición.

Cross-site Scripting (XSS) XSS es un exploit de seguridad que permite a un atacante injectar código malicioso de cliente en una página web. Este código es ejecutado por la víctima, y permite al atacante pasar los accesos de control, e impersonar al usuario. Este ataque tiene éxito si la aplicación web no tiene validación, o encoding.

Estos ataques ocurren cuando contenido entra en una aplicación web a través de una fuente no confianza, o contenido dinámico es enviado a un usuario sin estar validado. Suelen funcionar inyectando HTML en la página, en la que el usuario introduce credenciales, o extrayendo información de las cookies.

Para proteger contra este tipo de ataques, en la aplicación se llevan a cabo dos medidas, indirectamente. La primera, absolutamente todo el contenido de la estructura web llega desde el servidor front, y una sóla vez, y el resto de contenido no estructural llegará al cliente será desde una única fuente, el servidor. La segunda, React tiene mecanismos para proteger contra este tipo de ataques (se hablará de ello en la sección de seguridad de React).

SQLinjection Este tipo de ataques funciona ejecutando sentencias SQL contra una base de datos, pudiendo resultar en pérdida de datos, o extracción de los mismos.
El ORM de Django, junto con su estructura de datos para las queries, el Queryset, está protegido contra SQLinjection, ya que las queries están construidas con parametrización de queries, es decir, el código SQL ejecutado está separado de los parámetros que recibe la query, y estos parámetros a su vez son escapados en el driver de la base de datos.

Información delicada y buenas prácticas Adicionalmente a las vulnerabilidades típicas, hay una serie de medidas que se han tomado (o se podrían tomar), con respecto a buenas prácticas e información delicada.

- Las peticiones HTTP están delimitadas correctamente. Todas aquellas que precisen de atentificación, han sido configuradas de esa forma, y solo hay endpoints con GET ”abierto”, además, como dictaminan las buenas prácticas de seguridad, la implementación está hecha de forma que una petición de tipo

GET *nunca* va a modificar nada en la base de datos, ni crear, ni editar, ni borrar, solo leer.

- Información delicada (en este proyecto, contraseñas), no se envían nunca desde el servidor al cliente, por lo que un ataque que intercepte estas peticiones no va a encontrar más información que nombres de usuarios y fechas.

- Actualmente, debido al scope del proyecto, se ha desarrollado todo en local a modo de *Proof of Concept (POC)*, por lo que no hay necesidad de preocupación cuando el cliente manda información al servidor, pero en caso de ser lanzado alguna vez a producción, es *imperativo* que el sistema se hostee en HTTP**S**, de esta forma, las peticiones cliente-servidor, y las respuestas se enviarán encriptadas, por lo que nunca un Man-In-the-Middle va a poder interceptar información

- Aunque ya se ha mencionado previamente, es importante reiterar, las contraseñas *no* se guardan en texto plano, ya que se ha usado el modelo de Usuario y autentificación que vienen con Django y Django-Rest-Framework respectivamente (En estos temas, por lo general es muy mala idea querer reinventar la rueda)
  De hecho, una vez Django guarda las contraseñas, no hay forma de ver el texto plano, sólo existe la posibilidad de cambiar la contraseña, o validar si una contraseña es la correcta.

4.3. Front - React

En el portal hay cuatro ”vistas” (se define vista en este apartado como las pantallas disponibles, pero difiere del término vista que se menciona en la arquitectura MVC anteriormente), Landing, Login, Search y Profile. Están conectadas como muestra la figura 14. Cada una de ellas tiene una serie de componentes, cuyas relaciones se describen en la figura 32. Tras la explicación las diferentes vistas, se hace un breve repaso por los componentes exponiendo su utilidad.

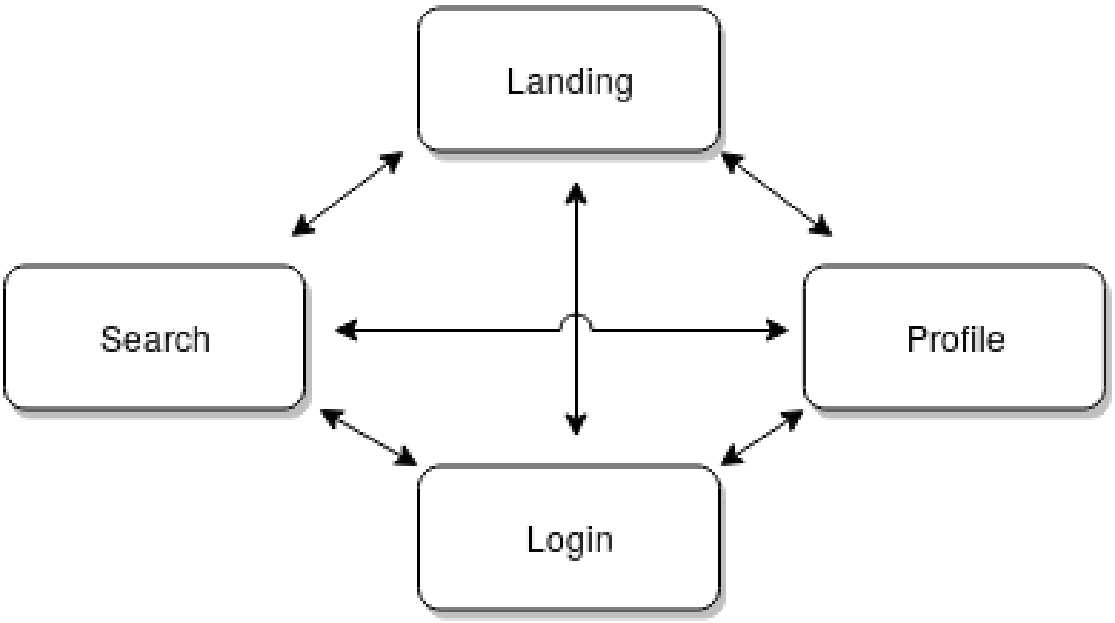

Figura 14: Diagrama de vistas

4.3.1. Header

Aunque no es una de las cuatro vistas mencionadas previamente, es necesario hacer una breve explicación del funcionamiento de este componente. Es un componente que todas las vistas (con excepción de Login) llaman, y sirve para tener una cabecera estándar en la página.

Adicionalmente provee de redirecciones en función de la situación, si se quiere hacer login, si se está logueado y se quiere salir de la sesión... Estas situaciones se explican en las siguientes secciones.

Provee en las diferentes vistas de un menú en función de la vista en la que se encuentre el usuario actualmente, la figura 15a enseña lo que ve un usuario estando en la vista Search, y la figura 15b muestra lo que se ve en ese menú al estar en la vista Profile. Desde la vista Search permite ir a Profile, y desde ambas permite modificar los datos del usuario, tras lo cual se abre un modal para editar (figura 30)

Por último, como es habitual en todas las páginas web de hoy en día, por lo que es un elemento de User eXperience (UX) importante, el logo del portal (*Nobe*), redirige a la vista Landing (el equivalente a Home, en esta aplicación).

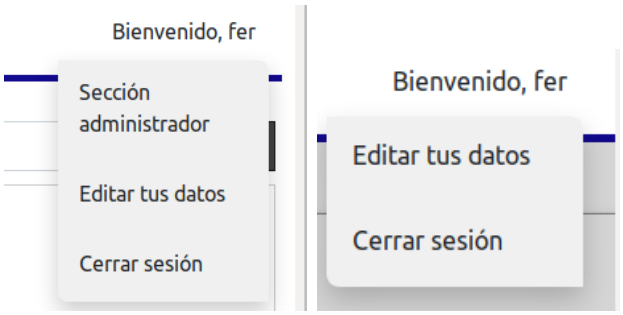

Figura 15: Menú de usuario tras hacer login en vista a)(izq.)Search, y b)(der.)Profile

4.3.2. Landing

Inicialmente, lo que ve cualquier usuario es la figura 16. El equivalente a Home en esta aplicación, es una User Interface (UI) muy sencilla, dirigida a un propósito en concreto: buscar. No hay distracciones de nada, y se presupone que la mayoría (estimación del 80/90 %) de los posibles usuarios que tenga una interfaz como ésta serán usuarios normales, y el resto administradores. Para la parte de administración, está el Login en la parte superior derecha. En general, se ha tratado de hacer la UX lo más sencilla posible.

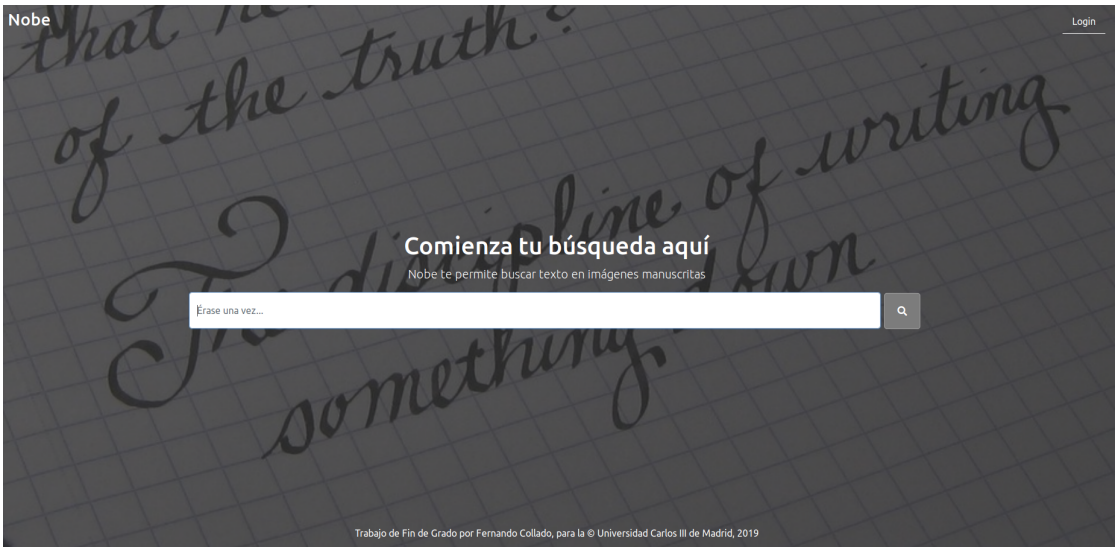

Figura 16: Landing de la interfaz

Además de tener un contraste claro en colores (UI), se hace una indicación breve de qué hace el sistema "*Nobe te permite buscar texto en imágenes manuscritas*, y un *placeholder* en la barra de búsqueda para dar lugar a la búsqueda.

Como se observa en la figura 16, Landing tiene conexión directa con las otras tres vistas. Desde Landing, si el usuario no está logueado todavía, puede

ir a Login. Si el usuario está logueado (tanto en esta sesión, como haber estado logueado previamente), al clickar en el texto que diga *Bienvenido, <usuario>*, irá directamente a su perfil. Por último, como es obvio, al realizar una búsqueda pasa a la vista Search.

4.3.3. Log in

La página de log in también se ha creado con la intención de ser lo más sencilla posible, una vez en esta página, o haces log in, o vuelves atrás donde estabas, tan sencillo como eso

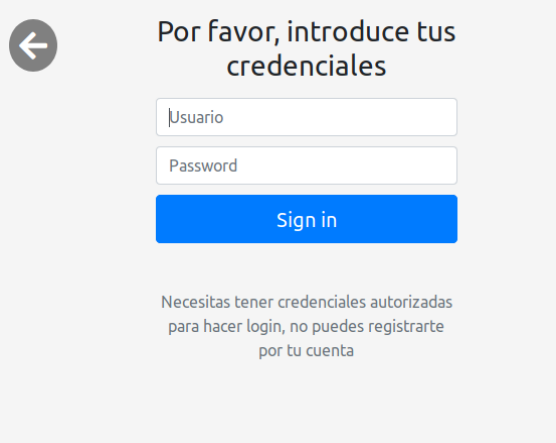

Figura 17: Login de la interfaz

Se puede llegar a esta página de dos formas, estando en Landing o en Search, y presionando *Login*, en la esquina superior derecha, o tras modificar los datos de contraseña de tu propia cuenta.
De la misma forma, desde Login se puede llegar a Profile, tras loguearse correctamente, y a Landing y Search, si se ha iniciado un intento de loguearse (por ejemplo, por error), y se desea volver atrás (con la flecha que se ve en la esquina superior izquierda en la figura 17).

4.3.4. Search

La página que muestra los resultados de búsqueda, y permite realizar búsquedas posteriores. Tiene tres partes, la cabecera, viene dada por el Header visto en esta sección, una columna de filtros y ordenación, y la sección de barra de búsqueda y resultados, como se puede ver en la figura 18.

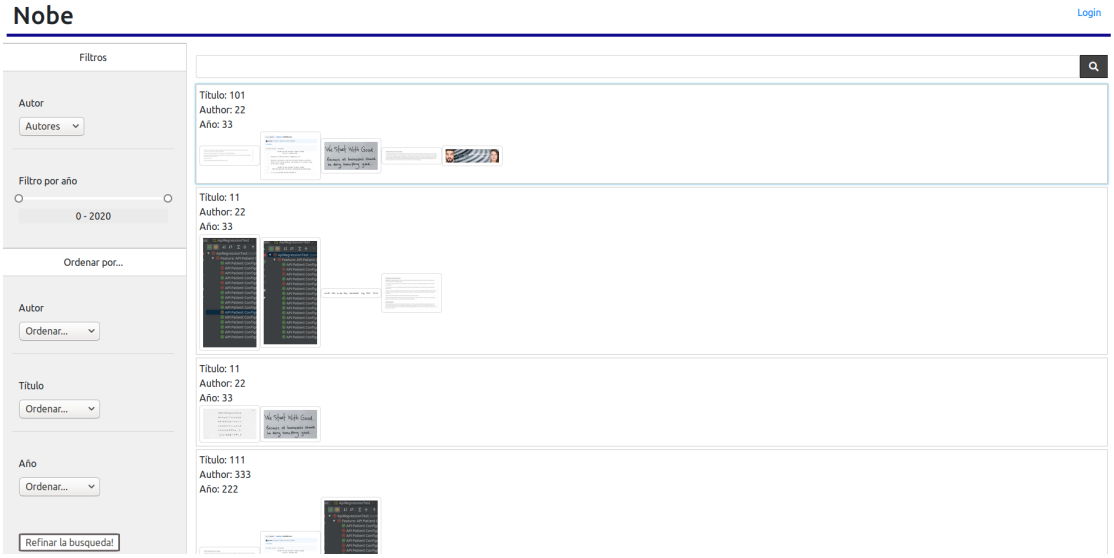

Figura 18: Vista Search de la interfaz

En la columna de la izquierda se presentan filtros por autor, y por año (a elegir entre dos fechas), y se da la posibilidad de order por autor, año, y título. Todas estas opciones son mutuamente compatibles, por ejemplo, se puede hacer ordenación por autor y año, a la vez que se filtra entre años 1950 a 1980, por autor Orson Scott, y se aplicará todas las opciones a la vez, se hará ordenación por año, y dentro de la ordenación por año, en cada año, estará ordenado por autor.
Para refinar la búsqueda con estos filtros y parámetros de ordenación hay dos opciones, la primera, clickando en el botón *Refinar la búsqueda*, y la segunda es realizar una búsqueda, con el mismo parámetro de texto que había antes de modificar los filtros, o cambiando el texto. Esto se ha hecho de cara a una mayor usabilidad, ya que no todos los usuarios buscarán un botón de refinamiento de búsqueda. También se ha tomado la decisión de no aplicar el filtro directamente ya que en ocasiones se desea realizar más de un filtro, u ordenación, y puede resultar poco amigable en el sentido de UX tener la página refrescando el contenido por cada parámetro que se cambie.

Una vez se ha hecho una búsqueda que tenga texto los resultados que se obtienen difieren de aquellas búsquedas que no tienen texto, a fin de ayudar al usuario a encontrar qué imagen exactamente es la que tiene el texto buscado. Esto se hace mediante el resaltado de la imagen o imágenes, que contienen el texto, como se puede ver en la figura 19

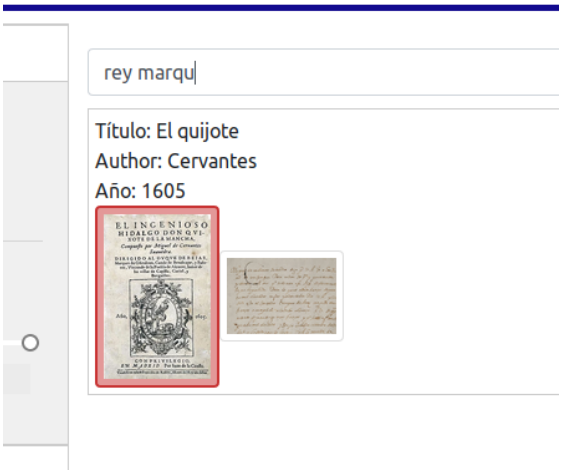

Figura 19: Imagen resaltada que contiene el texto buscado

El usuario puede clickar en las imágenes, y un *Lightbox* se abrirá mostrando dichas imágenes, empezando por la que se ha clickado. En caso de clickar en el registro y no en una imagen en concreto, el Lightbox se abrirá empezando en la primera imagen. Adicionalmente, las imágenes que antes se resaltaban tienen de nuevo un fondo distintivo de las demás para permitir diferenciarlas con facilidad. La imagen 20 muestra una imagen resaltada en el Lightbox, que se diferencia de la imagen 21, donde no se encuentra el texto buscado.

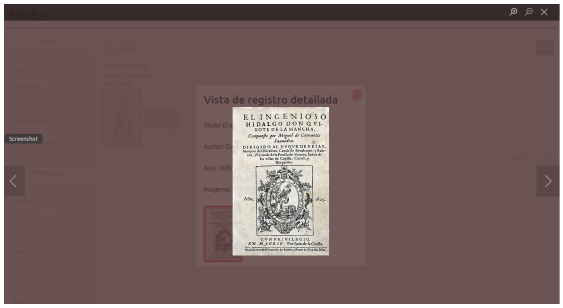

Figura 20: Lightbox abierto con una de las imágenes que contienen el texto encontrado

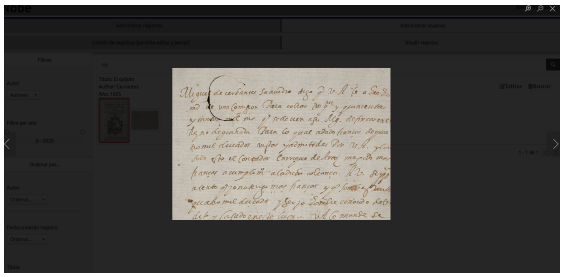

Figura 21: Lightbox abierto con una de las imágenes que **no** contienen el texto encontrado

En esta sección se ha puesto empeño en que todas las opciones que trae la página son visibles desde el principio, desde la columna de los filtros y ordenación, con el botón de refinar filtros de forma visible, pasando por los resultados encontrados que coinciden con el texto buscado, e incluyendo la paginación de los resultados, en la que se muestra la cantidad de resultados obtenidos en total, y los resultados actuales (por ejemplo, 6-10 de 17).

4.3.5. Profile

Por último, la sección Profile. Esta es la más compleja, ya que envuelve un mayor rango de funcionalidad, permite buscar, editar, eliminar y añadir registros y usuarios.

En la figura 22 se puede observar lo primero que un usuario ve tras registrarse. Está dividido en tres partes, el Header (explicado previamente), un selector de sección general y específica, y el contenido de la sección específica. Las secciones son, **Administrar Registros**, que contiene *Listado de Registros* y *Añadir registro*, **Administrar Usuarios**, que a su vez tiene *Listado de Usuarios* y *Añadir usuario*. Para navegar a una subsección en concreto, hay que estar primero en la sección correspondiente y luego seleccionar la subsección. En la figura 22 estaría en la sección Administrar Registros, subsección Listado de Registros. Esto se ha realizado de esta forma para que toda la funcionalidad esté disponible en dos clicks siempre.

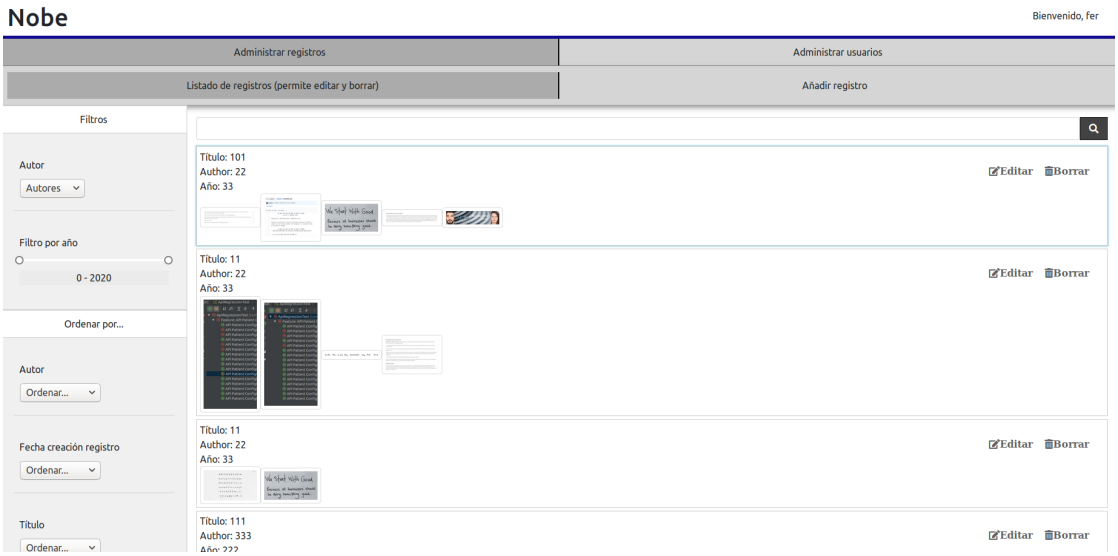

Figura 22: Profile en la interfaz

Administrar Registros

La subsección **Listado de regitros** es parecida a la página de Search, ya que permite buscar, aplicar filtros, ordenación... Y se muestran los elementos encontrados de la misma forma. El añadido aquí viene con la opción de editar y borrar. Al clickar en borrar un Modal es mostrado para confirmar el borrado del registro.

Al presionar en editar un modal con más opciones es mostrado, como se puede ver en la imagen 23. Este model se puede cerrar tanto en la esquina superior derecha (al igual que el resto de los modales del sistema), como con un botón de cerrar en el fondo del modal. Permite modifcar los metadatos del registro, Título, Autor, y Año de forma individual. Existen la posibilidad de añadir imágenes, presionando el botón *Añadir imágenes* dentro del modal observado en la imagen 23, con lo que se abre otro modal por encima, mostrado en la imagen 24, en el cual hay una Dropzone (zona para arrastrar y soltar, o clickar y seleccionar archivos) siguiendo el mismo estilo que al añadir un registro (se verá a continuación).

Figura 23: Modal editar registro

Figura 24: Modal añadir imágenes a un registro

Tras hacer click en *Editar imágenes*, se abre un Lightbox (imagen 25), y tenemos la posibilidad de modificar el texto reconocido por el sistema de ML, y se pueden borrar las imágenes. El proceso se mantiene mientras ese elemento no sea cerrado (es decir, si se modifica el texto de una imagen y se mueve a la siguiente, el texto modificado está todavía ahí), y no se guarda nada del progreso hasta presionar en *Guardar cambios y salir*. Una posibilidad podría haber sido la de guardar el progreso, y tras terminar de hacer cambios, volver a guardar, pero en este caso el usuario tendría que clickar en *Salir sin guardar cambios*, lo cual puede llevar a confusión, pues puede parecer que se están guardando los cambios en un borrador antes de confirmar. Por esta razón se mantiene el estado de las imágenes (hayan sido borradas, o el texto modificado), y simplemente se permite guardar una vez y salir.

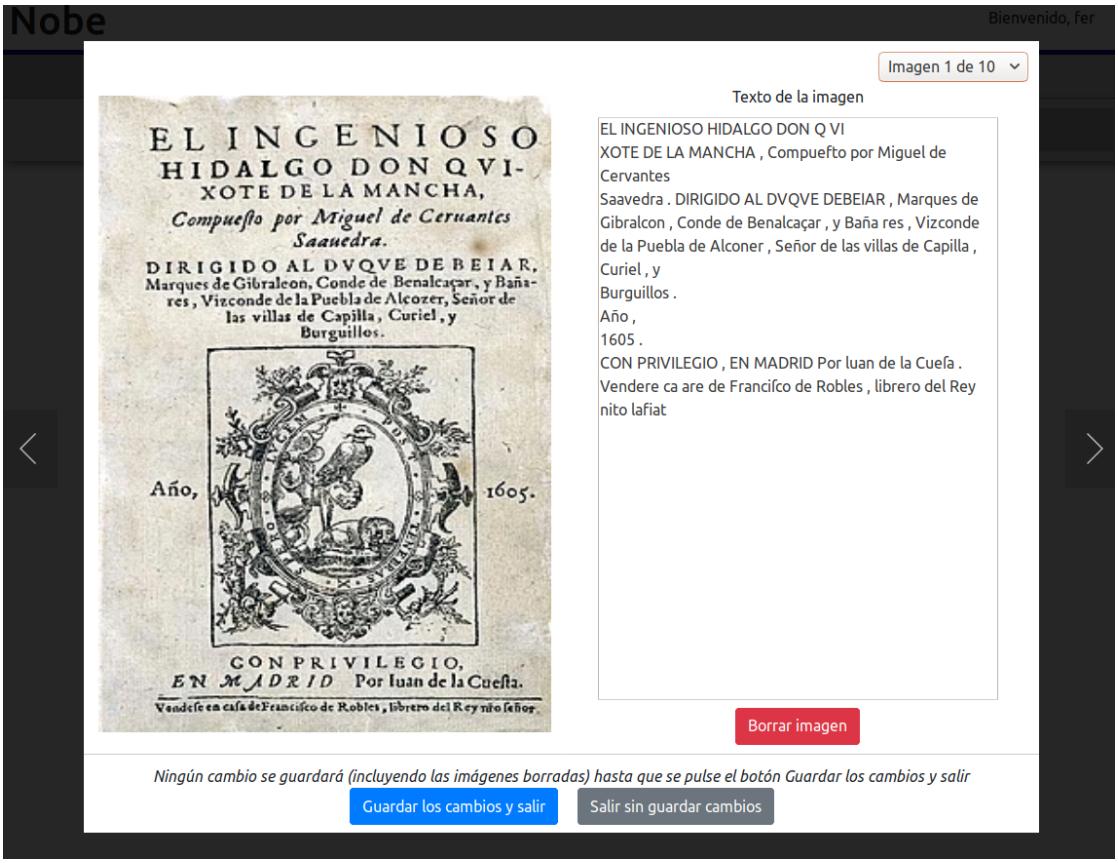

Figura 25: Modal editar imágenes de un registro

La subsección **Añadir registro** es una sencilla interfaz de usuario 26 en la que se presenta un formulario, con tres campos a rellenar, título, autor, y año (que tiene que ser formato: ser menor o igual que el año actual, y los años BC deben introducirse en negativo, 10BC ->-10). Todas estas restricciones son mostradas al usuario inmediatamente tras modificar datos, o si es necesario algún campo que no esté relleno, antes de hacer ninguna petición al back, un ejemplo de este mensaje de error es la imagen 28 Después, de nuevo un Dropzone para soltar las imágenes de forma cómoda, o buscarlas con un simple click. Se puede rectificar las imágenes añadidas dentro del propio Dropzone, un ejemplo de esto se puede ver en la imagen 27.

Figura 26: Formulario para añadir un registro

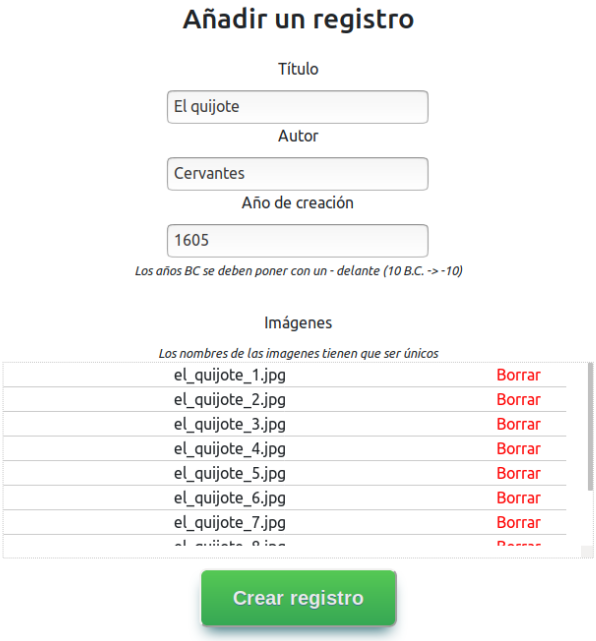

Figura 27: Formulario relleno para añadir un registro

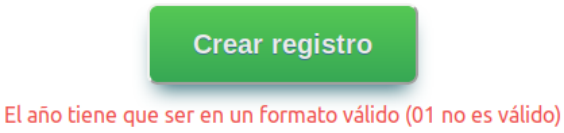

Figura 28: Mensaje de error por fecha incorrecta

Tras añadir imágenes, un modal es presentado al usuario preguntando si quiere revisar las imágenes, y si la respuesta es afirmativa, el modal mostrado en la imagen 25 es abierto, con los datos que devuelve el servidor sobre el registro que acaba de ser añadido.

Esta sección, *Administrar Registros*, ha sido sin duda la más difícil de hacer por la necesidad de elegir UI y UX que parezcan coherentes. Ante el requerimiento de hacer un sistema que permita revisar las imágenes añadidas y poder corregir texto que el algoritmo de ML haya podido confundir o no procesar, es necesario dar la opción, pero se ha considerado lógico añadir esa opción ya no solo como edición de imágenes en el listado de registros, también como opción tras añadir un registro. A su vez, elegir una forma de poder realizar cambios en el registro ha resultado complicado, hay diferentes opciones de visualización, y de realizar a cabo esta tarea, pero no todas parecían correctas. Se optó finalmente por separar las tres funcionalidades, modificar metadatos, añadir imágenes, y modificar el texto de una imágen o borrarla. En la sección de Testing se hará una conclusión sobre esta elección

Administrar Usuarios

Esta sección sigue la misma estructura que la sección anterior, contiene un **Listado de usuarios** y un **Añadir usuarios**, y ambos a su vez siguen una estructura similar, aunque menos compleja por razones obvias, que sus contrapartes.

En el listado de usuarios (figura 29) se puede buscar (en este caso buscará en camos username y email), y se puede ordenar por nombre de usuario, fecha de creación, y por email. Como con registros, permite editar (figura 30), y borrar todos los usuarios excepto a sí mismo. No se ha contemplado la posibilidad de permitir cerrar cuentas, ya que no se permite registrar libremente en este sistema. Respecto a esto se proponen un par de ideas en la sección de Trabajo futuro.

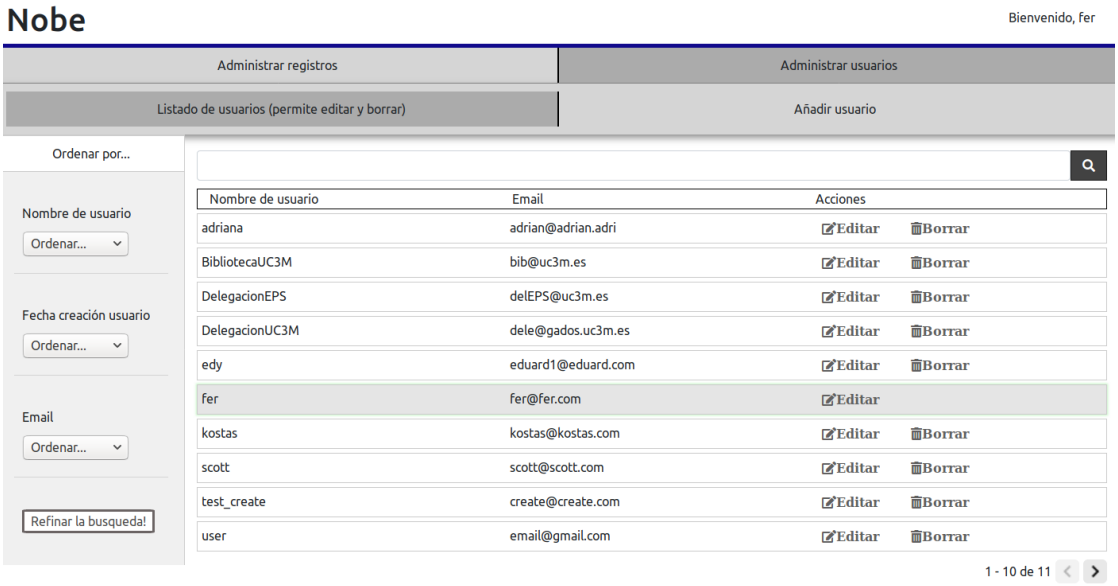

Figura 29: Listado de usuarios en Profile

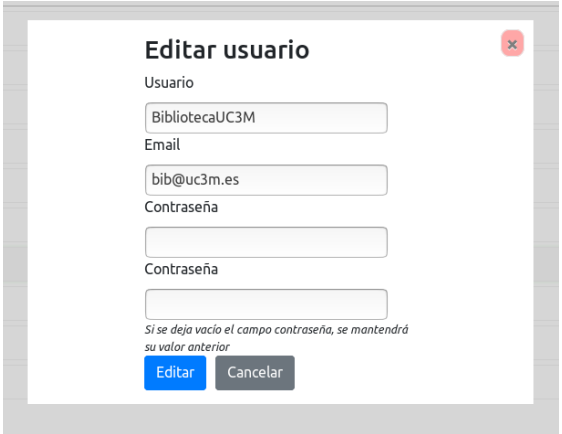

Figura 30: Modal para editar usuario

Por último, la subsección **Añadir usuario** de nuevo es una interfaz sencilla a fin de facilitar al usuario el uso de la plataforma. Como se puede ver en la imagen 31, es un formulario que consiste de nombre de usuario, email, y contraseña (de nuevo, en la sección de futuro trabajo se habla de una posible mejora, que no entraba dentro del alcance de este proyecto, a realizar respecto añadir usuarios).

Figura 31: Añadir un usuario

En esta sección se ha duplicado la estructura seguida en la sección anterior, *Administrar registros*, a fin de mantener una estructura común en la aplicación web, lo cual permite a los usuarios usarla con mayor facilidad. Cuando el comportamiento o la estructura es impredecible se corre el riesgo de que las funcionalidades implementadas no sean encontradas, o se desconozcan totalmente.

Como medida de estabilidad adicional, en el momento en el que un usuario cambia *su* constraseña, automáticamente se cierra su sesión (dada la implementación que se ha hecho en React, se elimina el token del almacenamiento local), y se le redirige a la página de Login, para que introduzca las nuevas credenciales.

4.3.6. Components

En la imagen 14 se puede ver la relación entre las diferentes vistas principales, y a continuación se muestra cómo se relacionan esas vistas con componentes en React. Como se ha mencionado previamente, los componentes *dumb*, aquellos que no tienen estado, se dedican sólo a renderizar información, y los componentes *smart*, que manejan estado, la forma de actualizarlo y validarlo, y pueden contener lógica adicional, como puede ser realizar peticiones al servidor para actualizar o borrar datos.

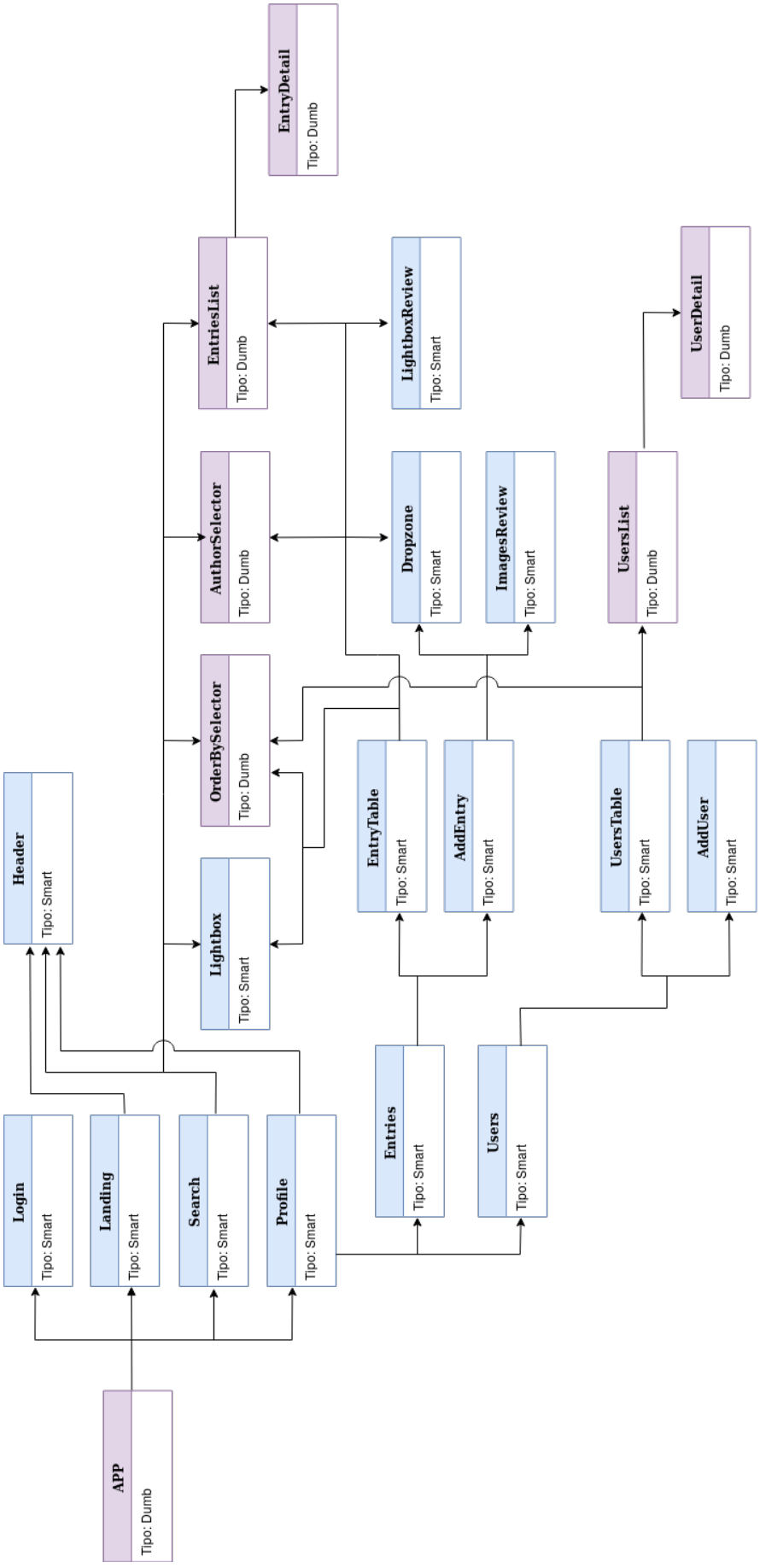

Figura 32: Diagrama de componentes del portal

De izquierda a derecha, y de arriba a abajo, los componentes son:

1. **APP**, el componente raíz de la aplicación. React funciona siempre teniendo este componente como raíz que llamará a otros componentes.

2. **Login**, Se encarga de la vista login, y de hacer las peticiones al servidor para el mismo propósito

3. **Landing**, se encarga de la vista Landing, y de realizar las peticiones de búsqueda en función del parámetro de búsqueda. Solo tiene un componente hijo, Header

4. **Search**, se encarga de la vista search, y mantiene en su estado todos los diferentes parámetros de búsqueda, filtros, ordenación y texto. Se encarga también de realizar las peticiones de búsqueda en función de estos parámetros. Tiene varios parámetros hijos, Header, Lightbox, OrderBySelector, AuthorSelector, y EntriesList.

5. **Profile**, se encarga de la vista Profile. Aquí se encuentran los componentes Header, Entries y Users. El estado que mantiene este componente es que sección está seleccionada, y en función de eso, mostrar Entries o Users. (Header se muestra siempre).

6. **Entries**, similar a Profile, tiene dos componentes hijos, EntryTable y AddEntry. El estado de este componente sirve para mostrar uno u otro en función de la subsección seleccionada.

7. **Users**, similar a Entries y Profile, tiene dos componentes hijos, UsersTable y Adduser, que muestra en función de cuál esté seleccionado.

8. **Header**. Este componente tiene dos funciones principales, la primera, mostrar la cabecera, que tiene el logo y el login (o el texto *Bienvenido <usuario>* si hay una sesión activa). La segunda función que tiene es la de controlar redirecciones a otras páginas según la situación, teniendo la posibilidad de ir a cualquiera de las otras páginas. En la imagen 33 se muestra el flujo de redirecciones

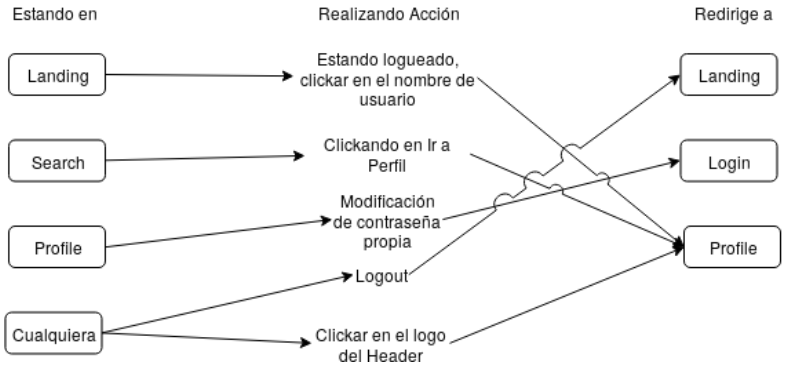

Figura 33: Redirecciones desde Header a vistas

9. **Lightbox**, componente que sirve para mostrar las imágenes en un carrousel infinito. Se ha utilizado un componente existente y disponible para descargar como paquete npm.

10. **OrderBySelector**, componente dumb al que se le pasa el valor del nombre a mostrar, y devuelve un selector con dos opciones. Se ha utilizado con propósito de reutilizar código

11. **AuthorSelector**, similar a OrderBySelector, dada una lista de autores muestra un selector con todos ellos.

12. **EntriesList**. Este componente dumb recibe una lista de registros, en caso de estar vacía, renderiza un mensaje informativo, pero si la lista contiene información, por cada elemento de la lista utiliza EntryDetail para mostrar la información del elemento.

13. **EntryDetail**, componente dumb, recibe la información de un registro, y renderiza los datos de Título, Autor, Año, y el listado de imágenes

14. **EntryTable**, el componente más complejo del sistema. Tiene como componentes hijos a LightBox, OrderBySelector, AuthorSelector, EntriesList, Dropzone, y LightboxReview. Sus cometidos son: mostrar la vista de búsqueda dentro del perfil, realizar búsquedas en función de los parámetros obtenidos, llamar al componente Lightbox cuando se clicka en uno de los registros, abrir los diferentes modales, el de edición del registro, y borrar registro, y todas las diferentes peticiones que esto conlleva.

15. **DropZone**, al igual que Lightbox, es un paquete disponible en NPM. Es un componente que se encarga de permitir soltar y arrastrar (o clickar en el componente) para recopilar imágenes.

16. **LightboxReview**. Con este componente se emula a LightBox, sirve para editar el texto de las imágenes de un registro. En el estado guarda el contenido del registro, las imágenes con su texto correspondiente, las imágenes modificadas, las imágenes a borrar, y el índice de la imagen actual. Además realiza la petición de actualización de estas imágenes.

17. **AddEntry** contiene el formulario para añadir imágenes. Usa componentes Dropzone e ImagesReview (que se explica a continuación). En su estado mantiene información del formulario e imágenes subidas, al igual que las modificaciones a las imágenes.

18. **ImagesReview** es un componente muy parecido a LightboxReview, pero existen algunas diferencias a la hora de mantener el estado, por lo que se duplicó el componente en vez de tener un solo componente que fuera muy complejo y difícil de mantener.

19. **UsersTable**, similarmente a EntriesTable, se encarga de mostrar la búsqueda de usuarios, manejar la ordenación de la búsqueda, y de la edición y borrado de usuarios. El estado que maneja es la información de los usuarios, y en caso de estar editando un usuario, la información temporal de edición antes de realizar la petición de actualización de ese usuario. Es por esto que también es complejo, aunque no tanto. Como componentes hijos tiene a OrderBySelector y UsersList.

20. **UsersList**, parecido a EntriesList, recibe una lista de usuarios. De estar vacía muestra un mensaje informativo, en caso contrario, por cada item de la lista llama a UserDetail con los datos del usuario. Es un componente dumb, por lo que no maneja estado alguno.

21. **UserDetail**, componente dumb que se encarga de renderizar cada una de las entradas de la lista de usuarios.

22. **AddUser**, similar a AddEntry, este componente tiene un formulario para añadir un usuario nuevo. El estado son los campos del formulario (username, email, contraseña, y la contraseña repetida), y se encarga de realizar la petición de crear un usuario nuevo.

4.4. Machine Learning

Como se verá más adelante, en la Sección 5.2, se ha elegido Google Vision API como método de reconocimiento de texto. Para usar esta API son necesarios varios

pasos, en diferentes etapas: configurar una cuenta en Google Cloud Platform, y permitir el uso de Vision API para esa cuenta. Los pasos a seguir se muestran en la Figura 34

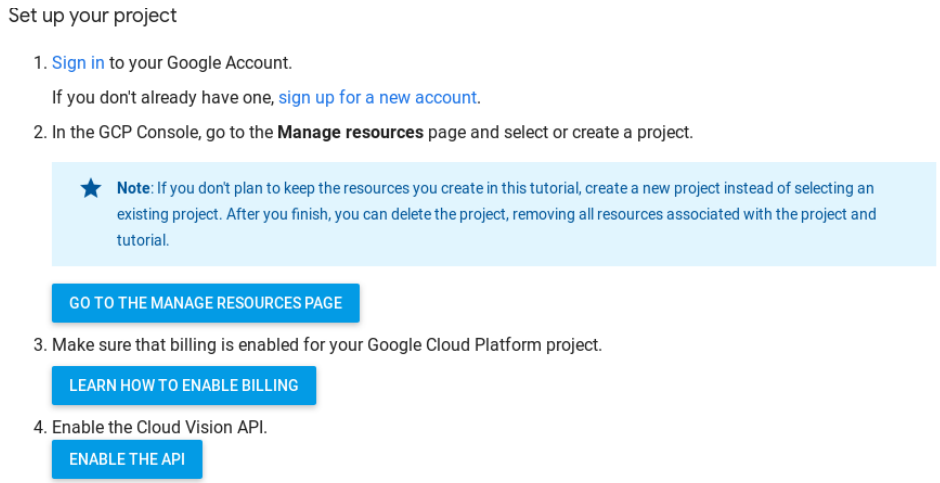

Figura 34: Pasos necesarios para usar Google API Vision

Tras crear la cuenta, se puede utilizar o bien llamadas ”puras” de API, en la cual se hacen *requests* a un recurso de Google con una API key (a modo de autentificación), y la imagen, o se puede usar una Service account, que es una forma de identificar una aplicación en el servicio de Google Cloud Platform, mediante el cual se tiene un mayor control sobre dicha aplicación. Debida a esta segunda razón, se ha optado por usar una Service account.

Para usar este tipo de cuenta, primero hay que seguir los pasos descritos en la figura 35. En el penúltimo paso, la creación de la cuenta en sí, hay que crear adicionalmente una clave, en formato JSON, que habrá que guardar en el servidor. Esto supone una medida de seguridad adicional al no tener explícitamente en el código ningún acceso a este servicio. Para que el servicio de google acceda a esta clave, se exporta en el servdior la clave, de la forma mostrada en código 18.

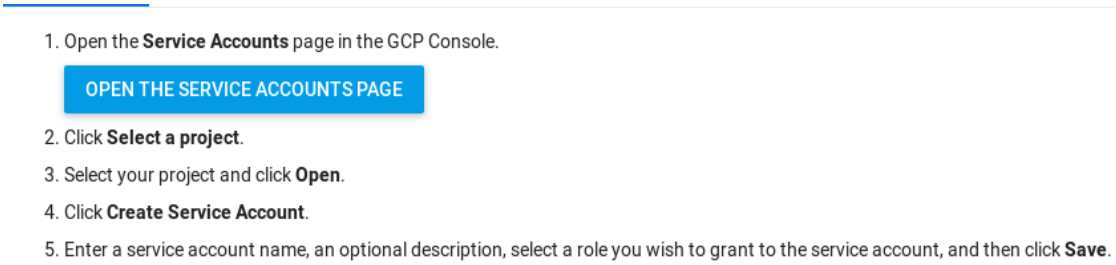

Figura 35: Pasos necesarios para usar una Service Account

Código 18: Exportar clave de acceso a Google Vision API

```
$ export GOOGLE_APPLICATION_CREDENTIALS=\textit{PATH_TO_KEY_FILE}
```

Una vez realizado el registro de nuestra aplicación en el servicio de Google Cloud Platform, con Google Vision API configurada, y las credenciales exportadas, solo falta implementar el código para hacer *requests* a la API. El código completo de la lógica se encuentra en el código 19.

Código 19: Uso de Vision API para detectar texto

```
from google.cloud import vision

client = vision.ImageAnnotatorClient()

def detect_document(image):
    """Detects document features in an image."""

    image = vision.types.Image(content=image)
    response = client.document_text_detection(image=image)
    text = ''

    for page in response.full_text_annotation.pages:
        for block in page.blocks:
            for paragraph in block.paragraphs:
                for word in paragraph.words:
                    text += ''.join([
                        symbol.text for symbol in word.symbols
                    ])

                    # Space between words
                    text += ' '

            # New line between paragraphs
            text += '\n'

    return text
```

Este código se encuentra dentro del servidor web, pero hace falta primero iniciar el cliente, de la forma mostrada en código 20. Este código se encuentra en el archivo *settings.py*. Se hace de esta forma ya que la conexión inicial a Google Cloud Platform a través de este método es algo lenta (uno o dos segundos), por lo que, al iniciarlo en el archivo *settings.py*, se ejecuta una vez el script, con lo

que se activa la llamada al cliente, y permite así el uso a cualquier parte del servidor Django con tan solo importar *from django.conf import settings*, y luego usar *settings.DETECTOR(<imagen>)*.

Código 20: Vision API en *settings.py*

```
from text_detection.detector import detect_document
DETECTOR = detect_document
```

5. Experimentación

En esta sección se recogen dos tipos de experimentación diferentes: entrevistas de usabilidad a usuarios primero, y después la experimentación realizada con los sitemas de ML para compararlos y elegir uno. A continuación se describen cada uno de estos tipos de experimentación.

5.1. Entrevistas de usabilidad

Se han realizado dos tandas de entrevistas, una primera tanda con dos personas más cualitativa, en un estado "temprano" del proyecto (cuando no estaba terminado 100 %), para empezar a testear la usabilidad del sistema y detectar fallos de forma temprana, y una segunda tanda más cuantitativa, en la que se entrevistó a cinco personas (ninguna de ellas la misma que las de la primera tanda), como prueba de usabilidad.

Primera tanda

En esta primera tanda, menos estandarizada que la segunda, se pidió a dos usuarios que realizaran tareas sencillas y rápidas, para comprobar la experiencia de usuario. Algunas de estas tareas implican aplicar filtros a una búsqueda, navegar por las páginas de una búsqueda, añadir una entrada, añadir un usuario, entre otras.
El feedback (tanto por parte de los usuarios directa o indirectamente, como el registrado por mi parte, el entrevistador, de lo observado) de esa sesión fue:

- Esperan poder utilizar teclas para realizar acciones como buscar (Enter), o salir de una vista detallada, o modal (Escape). Algunas de estas funcionalidades estaban implementadas, otras se han implementado después.
- La vista detallada del registro no convence a ninguno de los usuarios, ya que no aporta información extra. De cara al futuro, la intención era que la vista detallada tuviese más información que la proporcionada en la búsqueda, como por ejemplo, un resumen del documento. Pero ya que todavía no está contemplado, de momento se ha decidido eliminar ésta vista.
- Uno de los entrevistados pidió un botón "limpiar filtros". Se ha decidido no implementar esa funcionalidad debido a que es algo poco común en este tipo de sistemas, y puede llevar a confusión (¿debería vaciarse el campo de búsqueda? ¿Se considera el texto un filtro? Existen parámetros de ordenación, ¿se consideran filtros, o se debe hacer un botón extra para resetear la ordenación también?).

- Inicialmente, en la sección administración, los iconos de editar y borrar eran diferentes para registros y usuarios, diferentes tamaños, en el caso de registros solo eran botones y no tenían texto, y el color tenía un contraste mucho más alto. Ambos usuarios mencionaron algo respecto a esto, y se decidió hacer estándar los botones de editar y borrar, y con menos contraste.

- El campo de contraseña a la hora de añadir un usuario o editar un usuario no tenía una duplicación para que el usuario pudiera comprobarla, y no estaba oculto (al introducir la contraseña de un usuario nuevo, se podía leer). Ambos usuarios mencionaron que este comportamiento difería del normal para este tipo de casos, y fue rectificado al estado actual, dos campos de contraseña que se comprueba que concuerden, y los valores de dichos campos están ocultos.

- Tras crear un usuario, se eliminaba el valor del formulario HTML, y ya que estaban codificados para ser necesarios, aparecía un color rojo alrededor de los campos del formulario, lo cual hizo pensar a ambos usuarios que había ocurrido un error. Se ha modificado el código para que esto no pase.

- En los botones de confirmar la creación de registro y usuario, el texto del botón leía ”Confirmar crear registro”, y ”Confirmar crear usuario”. Uno de los usuarios mencionó que le parecía una forma extraña, como si fuese una repetición la frase en sí. Se decidió cambiar a ”Crear registro” y ”Crear usuario” respectivamente.

Otra parte útil de esta primera tanda fue encontrar un par de bugs en el portal, respecto a buscar usuarios y editar los datos de un usuario, que fueron solucionados de cara a la segunda tanda.

Segunda tanda

En esta segunda tanda el test se estandarizó, y se realiza en dos fases, tras una breve explicación inicial no muy detallada sobre lo que hace el sistema:

”Este sistema permite a un usuario (no registrado) realizar búsquedas. Estas búsquedas son en el texto de imágenes manuscritas. Adicionalmente, existe un portal administrador para usuarios registrados que permite manejar registros y usuarios”

Con esta explicación intencionalmente oscura con respecto a funcionalidades exactas se intenta saber si el sistema es intuitivo y si es necesaria una guía técnica. El test se divide en dos fases:

1. La primera fase del test es cualitativa. Se pide a los usuarios que realicen una serie de tareas, sin proporcionar ayuda a no ser que la pidan (y lleven un tiempo sin poder avanzar). En esta fase se han recogido hechos observados, y las opiniones y pensamientos del usuario. Las tareas que se pide realizar al usuario son:

Estando en vista	Tarea a realizar	Dificultad estimada
Landing	Buscar una palabra concreta	Baja
Search	Aplicar filtro entre años 1600 y 1800, y ordenar por autor la búsqueda anteriormente realizada	Media
Search	Autenticarse	Baja
Profile	Añadir un usuario	Media
Profile	Buscar una palabra en concreto, y añadir otra palabra al texto de la imagen que contiene la palabra buscada.	Alta
Profile	Añadir un registro, y modificar una de las imágenes subidas	Alta
Profile	Eliminar el mismo usuario previamente creado	Media

Cuadro 42: Test de experiencia de usuario

2. Por último, se les hace una serie de preguntas cuantiativas, en las que tienen que dar un valor del 1 al 5, siendo 1 *nada, poco, falso*, y 5 *mucho, todo, verdadero*. Las preguntas hechas son:

   a) Me gustaría usar el sistema

   b) He encontrado el sistema complejo (en este caso, complejo no se refiere a complicado)

   c) He encontrado el sistema fácil de usar

   d) Necesitaría soporte técnico para usar el sistema

   e) He encontrado incosistencias en el sistema

   f) Necesito aprender mucho para usar el sistema

Para esta segunda fase se han elegido deliberadamente a usuarios de diferentes transfondo en cuanto a estudios, y edad. El feedback agregado de las entrevistas es (cada número corresponde con la tarea cualitativa correspondiente en el Cuadro 42):

1. *Buscar una palabra concreta:* Esta tarea de dificultad baja solo ha tenido como feedback por parte de los usuarios ”intuitivo”.

2. *Aplicar filtro:* Un usuario lo encontró fácil de hacer, otro comentó que no es la forma común de hacerlo, ya que por lo general solo hay una forma de ordenar, pero le resultó intuitivo. El tercer usuario encontró dificultad para seleccionar los años, y seleccionó un parámetro de ordenación incorrecto, el cuarto simplemente mencionó que la parte visual debería ser ligeramente más grande, y el último entrevistado mencionó que la parte de ordenación "da la impresión de que va a ser para una etiqueta", sin dar más explicación al respecto. Aun así consiguó realizar correctamente la tarea

3. *Autenticarse:* Todos los usuarios han encontrado de forma inmediata el botón de *Login*, ninguno ha tenido ningún problema o comentario negativo respecto a esta tarea.

4. *Añadir un usuario:* Esta tarea tiene como finalidad comprobar si el sistema de sección y subsecciones en la vista Profile funciona de forma intuitiva o no. Para ejecutar esta tarea los usuarios tiene que seleccionar la sección Administrar usuarios, y después la subsección Añadir usuario. Hay un mix de resultados en este caso, teniendo usuarios que lo encuentra confuso al princiio, pero muy intuitivo una vez se dió cuenta de cómo funcionaba, otro no ha visto la diferencia de las secciones, dos han encontrado muy rápido la pantalla de añadir un usuario, y el último sabe encontrar rápido la subsección, pero cuando se quiere mover entre las subsecciones lo encuentra confuso.
   Adicionalmente, dos usuarios en concreto procedieron a comprobar que el registro hubiese sido creado, a pesar de recibir un mensaje de confirmación de creación.

5. *Buscar una palabra en concreto, y añadir otra palabra al texto de la imagen:* Esta tarea en concreto es la más difícil de todas, ya que envuelve varias subtareas y funcionalidades "ocultas". La primera de esas funcionalidades ocultas es el resaltaado de una imagen que contiene el texto buscado. La segunda, ya que los usuarios no conocen a fondo el sistema, la posibilidad de editar el texto de una imagen (y dentro del Lightbox de edición, poder navegar libremente entre las imágenes). La combinación de estas dos funcionalidades es lo que se quería comprobar con esta tarea. Solo un usuario se ha dado cuenta de que la imagen resaltada es la que contenía el texto, y sólo un usuario (no el mismo) se dió cuenta de que tenía que navegar hasta la imagen que contenía el texto, el resto modificó la primera imagen y salió guardando los cambios.
   Otros problemas encontrados en esta tarea fueron: dificultad para llegar

hasta la opción de editar las imágenes y abrir el Lightbox, algunos usuarios pensaron que la imagen resaltada significaba que la imagen estaba seleccionada, y que el Lightbox se abría con esa imagen en concreto (ergo, tampoco se dieron cuenta de que el Lightbox permite moverse entre imágenes).

6. *Añadir un registro, y modificar una de las imágenes:* Esta tarea fue realizada con más o menos un alto grado de éxito entre los usuarios, recibiendo como feedback que es intuitivo, facilidad para encontrar dónde tiene que ir, y como notas negativas, que el borde del Dropzone podría ser más visible, y (por parte de un usuario), falta de información por parte del sistema mientras está procesando la información de las imágenes.

7. *Eliminar el mismo usuario previamente creado:* Los entrevistados han encontrado en general la última tarea sencilla de hacer, ya que estaban hechos al sistema de secciones y subsecciones. Ningún usuario ha encontrado problema o ha realizado la tarea de forma lenta.

Como se puede observar, la experiencia de usuario no ha sido perfecta. Esto indica que es necesario re-pensar la interfaz, o implementar mejoras en el sistema. Varios usuarios mencionaron a lo largo del proceso que estaban seguros que si tuvieran un tutorial, o supiesen más de la plataforma desde el principio puede que no tuviesen problemas. En particular la opción de editar imágenes es la más problemática, junto con las secciones y subsecciones. En cuanto a éste segundo problema se dio sobre todo al principio, ya que los usuarios se adaptaron rápido a ella, por lo que no debería ser particularmente grave o difícil de corregir.

Respecto a los resultados de las preguntas cuantitativas, se pueden observar en el cuatro 43.

Pregunta	1	2	3	4	5
1	0	0	0	2/5	3/5
2	0	1/5	3/5	0	1/5
3	0	0	0	4/5	1/5
4	1/5	3/5	1/5	0	0
5	5/5	0	0	0	0
6	4/5	1/5	0	0	0

Cuadro 43: Resultados test cuantitativo a usuarios

Como se puede ver por los resultados, una gran parte de la gente le ha encontrado una utilidad personal (diferentes usuarios me comentaron para qué usarían

ellos personalmente un sistema como éste), y han encontrado el sistema complejo, pero no particularmente complicado, ya que ningún usuario dio menos de un 4 en la pregunta de facilidad de uso.

Esto también se ve reflejado en el resultado del soporte técnico, donde solo una persona se posiciona neutral con respecto a si necesitaría soporte técnico o no, mientras que las otras cuatro responden que o bien un poco, o directamente nada de soporte sería necesario para ellas.

Por último, ningún usuario encontró incosistencias en el sistema, y solo un usuario respondió diferente de (el equivalente de 1 a 5 a) "falso" en la pregunta de si necesitaría aprender mucho para aprenderlo

Tras observar el resultado de ambos tests, se puede concluir que la experiencia de usuario es generalmente buena, y aceptada por los usuarios entrevistados, pero todavía quedan cosas por mejorar.

5.2. Machine Learning

Hoy en día existen numerosas formas de conseguir cumplir objetivos en tareas de Machine Learning, pero siempre hay un problema: recursos. Cualquier persona puede entrenar un modelo en su casa, con un portátil de hace 10 años, pero eso no significa que el resultado vaya a ser bueno. Modelos precisos requieren de muchos datos, y repeticiones (y un ajuste preciso de hiperparámetros), y repeticiones significan tiempo. Debido a ésto, se ha optado por comprobar directamente el resultado de APIs de compañías que tienen tiempo y dinero para entrenar modelos de forma precisa Además, entrenar un modelo buscando ajustar hiperparámetros de forma muy precisa queda fuera del alcance de este trabajo.

Para hacer una comparativa de estos modelos, se van a usar métricas de similitud de texto. Estas medidas tienen tres categorías [Gomaa and Fahmy, 2013]:

- **String-based**, donde la similitud se basa en secuencias string y composición de caracteres. Los algoritmos en esta catergoría se dividen a su vez en dos, character-based, y term-based. Algunos de estos algoritmos son Longest Common Substring, Damerau-Levenshtein, o N-gram para character based, y block distance, cosine distance, o Jaccard similarity para term-based.
- **Corpus-based**, donde la similitud está basada en el significado de las palabras. Se apoya en dos principios, contener un Corpus, una extensa

cantidad de textos que sustentan de significado a esta categoría, y (por lo general) de contexto. En algunas técnicas el contexto significa mirar en las palabras cercanas de la misma frase, en otras se basa en comparar la posición de la palabra en el texto. Algunos algoritmos de esta categoría son Analagía de Hipeserespacio a Lenguaje (HAL), Análisis Semántico Latente (LSA), o Normalized Google Distance.

- **Knowledge-based**, similarmente a corpus-based, la similitud también viene determinada por el significado, pero difiere de este método por la forma de obtener la similitud.En corpus based se obtiene de un corpus, en knowledge-based se obtiene de de redes semánticas, como Wordnet.

Debido a la naturaleza de la tarea, el significado de la palabra y su contexto no van a aportar nada a la hora de comprobar el correcto reconocimiento. La posición de las palabras, y el correcto número de ocurrencias es lo que sí va a ser útil. Por lo tanto se hace uso de cuatro técnicas *String-based*:

1. Damerau-Levenshtein: Es la distancia entre dos Strings medida como el número mínimo de operaciones para transformar un string en otro, donde una operación es añadir, eliminar, sustitución de un caracter, o transposición entre dos caracteres. Por ejemplo, la palabra *Domingo* y la palabra *Doming* tienen una distancia Damerau-Levenshtein de 1, mientras que las palabras *vuelo* y *huevo* es de 2. Es una buena medida para saber si el reconocimiento es muy aproximado a la realidad. Pero tiene como inconveniente que si el texto está repartido por la imagen, diferentes técnicas pueden reconocer el texto en distinto orden.

2. Damerau-Levenshtein Normalizado: de la misma forma que la Dameru-Levenshtein, pero normalizada, por lo que representa el portencaje de cambios necesitados respecto a la cantidad total de caracteres en el texto. Es necesario contextualizar el resultado de la distancia Damerau-Levenshtein con su versión normalizada, ya que por cómo funciona el algoritmo, simplemente muestra la distancia, pero no el impacto de dicha distancia. Un texto de longitud 100, que necesite 5 cambios, no tiene la misma significancia que un texto de longitud 7, que necesite 5 cambios.

3. Jaccard Similarity: Se calcula como el número de palabras compartidas entre ambos strings, teniendo en cuenta solo ocurrencias únicas de cada palabra. Por ejemplo, de la frase "Él vino, y tomó una copa de vino", lo que este algoritmo tendría en cuenta para calcular la similitud con otro string sería

["Él", "vino", ƶ", "tomó", ünа", çopa", "de"]. Si bien se pierde el factor similitud exacta entre los strings, muchas de las palabras repetidas en un texto van a ser preposiciones, las cuales carecen de significado real. La palabra "de", que puede aparecer diez o quince veces por texto, carece de la importancia que otras palabras tienen.

4. Cosine similarity: Mide la similitud entre dos vectores, mediante el ángulo que proyectan entre los vectores. En el ámbito de comparación de texto, los vectores están formados por la frecuencia de cada palabra. En el ejemplo anterior, *"Él vino, y tomó una copa de vino"*, el vector resultante de calcular el vector es: [1, 1, 1, 1, 2, 1], para las palabras ['copa', 'de', 'tomó', 'una', 'vino', 'él'] mostrando así un valor de 1 para todas las palabras, excepto 2 para vino. Adicionalmente, este algoritmo ignora signos de puntuación y palabras de longitud 1, por lo que "y" no se ha tenido en cuenta.

Para los experimentos, se han utilizado 5 imágenes, y en cada una de ellas se ha medido la similitud con las 3 métricas mencionadas anteriormente entre el texto reconocido por el algoritmo de ML, y el texto real. Las imágenes usadas son las que se muestran en las Figuras 36, 37, 38, 39 y 40. La primera [5] tiene como finalidad comprobar la correcta extracción de letras y símbolos, la segunda es la misma que la primera pero con ruido *salt and pepper* añadido. La tercera, un texto escrito con fuente máquina, para comprobar que este tipo de texto se sigue reconociendo de forma correcta. La cuarta [6] es una carta manuscrita, en castellano, con tonos de castellano antiguo, y por último, es una frase sacada de la base de datos IAM [Marti and Bunke, 2002], muy comunmente usada en tareas de reconocimiento de texto mediante técnicas ML.

Figura 36: Imagen de prueba de reconocimiento de texto, es la letra escrita a mano de John Lennon, extraída como fuente.

[5]Imagen extraída de https://www.dezeen.com/2018/04/10/kurt-cobain-david-bowie-john-lennon-handwriting-typeface/

[6]Extraída de https://www.europeana.eu/portal/en/record/2022704/lod_oai_ciconia_gobex_es_10545_ent1.html

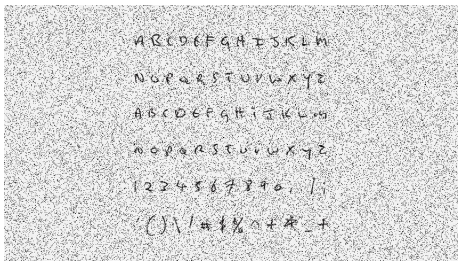

Figura 37: Imagen de prueba de reconocimiento de texto, con un filtro salt-pepper aplicado

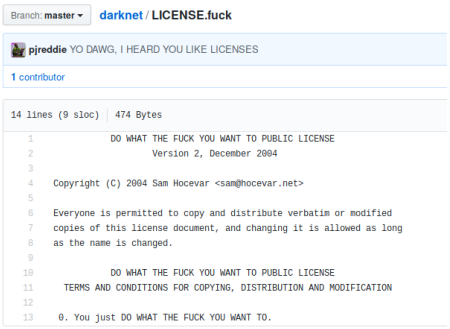

Figura 38: Imagen de prueba de reconocimiento de texto, con texto a máquina

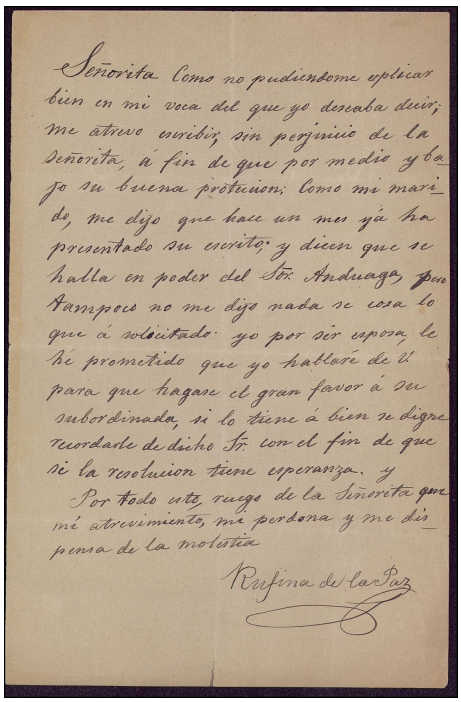

Figura 39: Imagen de prueba de reconocimiento de texto

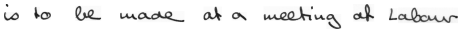

Figura 40: Imagen de prueba de reconocimiento de texto, extraída de la base de datos de imágenes IAM

Usando **Jaccard similarity** para comparar obtenemos los resultados que se muestran en el cuadro 44

Jaccard Similarity					
	36	37	38	39	40
Google	0.0	0.0	0.546	0.425	0.889
Azure	0.0	NA	0.427	0.31	0.7

Cuadro 44: Resultados de Jaccard Similarity

La implementación hecha provee de valores de similitud entre 0 y 1, cuanto más cerca del 0, menos parecido, y cuanto más cerca del 1, más parecido. Como podemos observar, para las imágenes 38, 39 y 40, Google API tiene mejor resultado en todos los aspectos. En este caso, para la imagen 36, el resultado es 0 ya que se ha juntado todo el texto de esa imagen en concreto para analizar mejor el correcto reconocimiento de caracteres. Adicionalmente, Azure ha sido incapaz de reconocer ningún texto en la imagen con ruido.

Similarmente a Jaccard Similarity, la implementación de **cosine similarity** (Cuatro 45) provee de valores entre 0 y 2 (porque calcula el dot product de una matriz de dimensión 2x2, cuyos valores están comprendidos entre 0 y 1). De nuevo, cuanto mayor sea el valor de cosine similarity, mayor similitud existe entre los textos. Usando Cosine similarity, se observan resultados similares a Jaccard, donde Google obtiene mejores resultados en todas las imágenes, excepto las dos excepciones mencionadas anteriormente.

Cosine Similarity					
	36	37	38	39	40
Google	0.0	0.0	1.981	1.793	1.789
Azure	0.0	NA	1.781	1.559	1.604

Cuadro 45: Resultados de Cosine Similarity

Por último, usando la distancia de **Damerau-Levenshtein** (y su versión normalizada), se observan los resultados mostrados en el cuadro 46.

Damerau-Levenshtein					
	36	37	38	39	40
Google	3	42	60	89	5
Google (normalizada)	0.0385	0.857	0.103	0.130	0.125
Azure	6	NA	42	106	4
Azure (Normalizada)	0.0759	NA	0.0725	0.156	0.108

Cuadro 46: Resultados de Damerau-Levenshtein

Los resultados de la primera fila de cada técnica muestran la distancia absoluta, y la segunda línea muestran la normalizada. Esto quiere decir que, para Google y la primera imagen, hay una distancia de 3, que se traduce a 0.0385 normalizada (o lo que es lo mismo, 3.85 % de diferencia con el texto original). En este caso, los resultados son más parejos en cuando a "victorias", ya que Google obtiene un mejor resultado en dos imágenes, mientras que Azure es mejor en otras dos imágenes. Pero, aún así, Google sale en cabeza en esta prueba por dos razones principales, la primera es que la imagen 39 es la que más se asemeja a lo que se puede esperar recibir en este sistema, y la segunda, aunque con un error muy alto, reconoce texto en la imagen con filtro salt-pepper, mientras que Azure no ha podido reconocer nada.

Dados los experimentos, se considera que Google obtiene mejores y se utilizará la API de Google. Adicionalmente, ésta es más directa de usar, ya que la API de Azure, tras enviar la imagen, devuelve otra dirección donde, una vez terminado el procesado, estará el resultado, lo cual significa que hay que hacer consecutivas requests a esta segunda URL, hasta que el recurso esté listo. Esto añade un *overhead* al sistema que no se ha considerado adecuado.

6. Organización del proyecto

Todo el trabajo explicado en la tésis corresponde a la implementación y estudio de viabilidad del proyecto desarrollado, que es posible gracias a una planificación previa. En esta sección se muestra la planificación, el contexto socioeconómico y marco regulador del proyecto

6.1. Planificación

Hay dos aspectos claves a la hora de planificar un proyecto: la planificación de tiempo, y la planificación monetaria, es decir, el presupuesto.

6.1.1. Planificación de tiempo

Para un proyecto de este tamaño es necesario planificar en varias etapas el proyecto, ya que existe tanto diseño y desarrollo del proyecto, como estudio y evaluación de las herramientas a utilizar. Las fases en las que se divide el proyecto son:

- **Análisis**: En esta primera fase se establece qué funcionalidades tendrá el sistema. Se especificarán los casos de uso y se recogerán los requisitos para el correcto funcionamiento del proyecto.
- **Investigación**: Se evalúa qué tecnologías se deben utilizar, a la par que se estudia el estado del arte.
- **Diseño**: Comprende el diseño de la arquitectura, las diferentes partes del sistema, cómo se van a comunicar los diferentes servidores (y cliente), y se establecen ”contratos” entre front y back (siendo un ”contrato” un acuerdo entre ambas partes sobre los endpoints, y cómo se tienen que enviar y recibir los datos en dichos endpoints)
- **Implementación**: Desarrollo de las distintas partes del sistema.
- **Testing**: Se evalúa mediante entrevistas de usuario el sistema
- **Documentación**: se redacta la memoria en base a los resultados obtenidos en las fases anteriores

En la figura 41 se muestra un diagrama de Gantt de las fasses del proyecto y su duración

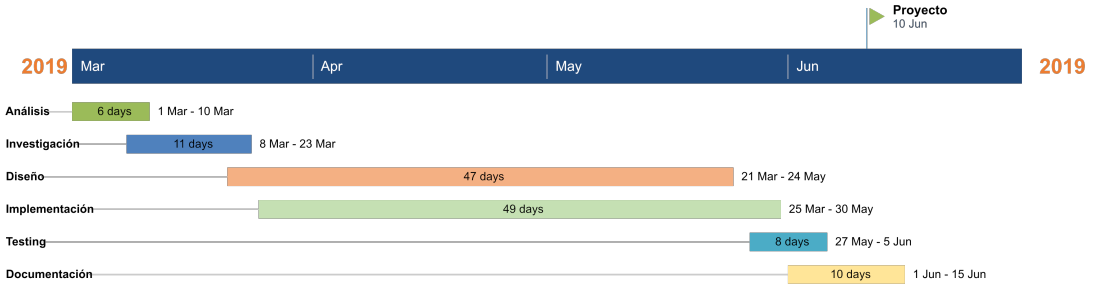

Figura 41: Diagrama Gantt con la planificación del proyecto

Como se puede observar, la parte de diseño e implementación van muy parejas en el tiempo, ya que, una vez pasada la fase de análisis e investigación, se ha seguido la metodología SCRUM, que es una metología ágil basada en sprints cortos, en los que se desarrolla un MVP (Minimum Viable Product) en cada sprint, o lo antes posible. Siguiendo esta metodología, para el desarrollo se han hecho sprints en los que diseño (principalmente la parte de contratos) ha ido de la mano junto con desarrollo.

6.1.2. Control de versiones

Para manejar el código del proyecto es necesario usar control de versiones (Version Control System VCS). El sistema elegido ha sido Git, un VCS extendido que permite mantener la pista de los cambios realizados. Por la forma en la que Git funciona, nos permite mantener todo el código en un repositorio, incluyendo dinámicas de desarrollo de código como puede ser crear ”ramas” (branches) de código, para diferentes funcionalidades y luego juntarlas en la rama principal, permite recibir Pull Requests, cambios que se quieren hacer, para que pasen una supervisión, entre muchas de sus funcionalidades. Una de las grandes ventajas de este sistema, adicionalmente a su intención inherente de mantener versiones de código, es ser usado como medida de seguridad ante pérdidas de datos en local. Al estar el repositorio en un servidor remoto, si el proyecto es desarrollado por una sola persona, puede subir el código a Git a modo de backup, o para desarrollar en diferentes dispositivos.

6.1.3. Presupuesto

Se definen tres posiciones para este proyecto, Jefe de proyecto, que se encarga de diseño (conjuntamente con Analista programador), planificación, y documentación. Analista programador, que se encarga de evaluar y estudiar herramientas a usar, técnicas que se deben utilizar, diseño, e implementación. Por último, pro-

gramador, que se encarga solo de implementación.

El tiempo trabajado dedicado por cada posición se recoge en la tabla 48, tras hacer un cómputo de horas por posición en la tabla 47.

Mes	Jefe de proyecto	Analista	Programador	Horas
Marzo	20	40	20	80
Abril	20	40	20	80
Mayo	65	0	65	130
Junio	30	0	30	60
Total	135	80	135	350

Cuadro 47: Tiempo por posición planificado

Posición	€/ Hora	Horas invertidas	Coste
Jefe de proyecto	35	135	4725 €
Analista programador	20	80	1600 €
Programador	15	135	2025 €
Total			8350 €

Cuadro 48: Coste de personal

Además del coste de personal, es importante computar el precio del hardware y software usado durante el proyecto. El tiempo de amortización es 48 meses, y el proyecto tiene una duración de 4 meses. El valor amortizado se calcula con la siguiente fórmula

$$Valor\ amortizado = \frac{Meses\ del\ proyecto}{Periodo\ de\ depreciación} \times\ Coste\ del\ equipo$$

Item	Precio total	Depreciación (en meses)	Valor amortizado
Lenovo IdeaPad 430U	799.99€	48	66.67€
Pycharm	0.00€	0	0.00€
VSCode	0.00€	0	0.00€
Ubuntu	0.00€	0	0.00€
Github	0.00€	0	0.00€
Total			66.67 €

Cuadro 49: Costes de equipamiento

Usando los cotes de personal y equipamiento, más IVA

El presupuesto toal de este trabajo asciende a la cantidad de DIEZ MIL CIENTO OCHENTA Y CUATRO CON DIECISEIS euros (10.184.16€)

Concept	Total
Costes de personal	8350.00 €
Costes de equipamiento	66.67 €
IVA (21 %)	1767.49 €
Coste de Proyecto	**10184.16€**

Cuadro 50: Coste total del proyecto

Es importante mencionar que la API de Google Visio establece un precio por uso, a partir de ciertos usos, pero no es parte estrictamente del presupuesto de desarrollo, entraría parte de un contrato con un hipotético cliente, en el que el valor total del contrato correspondería con el presupuesto del proyecto, una previsión del 10 % para imprevistos, y un valor condicional que está sujeto a uso.

6.2. Marco regulador

Dentro del marco regulador, solo hay hay dos que afectanal proyecto. La primera, la Reglamento General Protección de Datos (RGPD), debido al manejo de datos de usuario, y derechos de autor, debido a que permite subir contenido al servidor.

6.2.1. Protección de datos

Dentro de la protección de datos, desde el 25 de Mayo de 2018 se aplica una normativa general europea, Reglamento General de Protección de Datos (RGPD), que permite especificaciones locales en ciertos puntos del reglamento. De ahí surge en España la Ley Orgánica de Protección de Datos y Garantía de los Derechos Digitales (LOPDGDD). La LOPDGDD [BOE, 2018] sustitute a su vez la Ley Orgánica de Protección de Datos (LOPD).

La RGPD contempla varios niveles, pero el principal es, la RGPD aplica [AEPD, 2018] si los datos manejados permiten identificar con cualquier valor almacenado. En el caso de este trabajo, se hace uso de un nombre de usuario y del email, que es considerada información suficiente como para identificar, por lo que la RGPD aplica, y se deben seguir las directrices.

Lo primero que marca la RGPD es una evalución de impacto (EIPD) en la que se determinará el nivel de los datos almacenados y el impacto en la privacidad por almacenar estos datos. En este caso, el nivel está determinado como bajo, ya que no se trata con información sensible (aquellos datos que identifican más allá de

una persona física, como por ejemplo raza, orientación sexual...), no se adopta ninguna decisión automatizada (no se vende ni presta información a terceros), ni entra dentro de la categoría de transferencias internacionales.

Estos tres criterios mencionados previamente pertenecen a la parte de *consentimiento explícito* recogido en la RGPD, existe además *consentimiento implícito*, el cual dictamina que un usuario tiene que ser creado por petición, y la información que accede a ser procesada debe ser elegida por el propio usuario (no se permite el uso de datos preseleccionados, como por ejemplo, las cajas de opciones para acceder a recibir publicidad). En el caso de esta aplicación, el usuario es creado a petición expresa (y una vez instaurado el sistema más allá de esta versión POC, el sistema de mailing mandaría un correo al usuario para que confirme su registro).

Tanto la RGPD, como la LOPDGDD contemplan una serie de derechos para cada usuario. Derecho de acceso, que está dado por defecto en la plataforma se permite acceder a todos los datos personales. Derecho de rectificación, de nuevo, permitido por la propia plataforma; derecho de cancelación, con el que se pide la supresión de datos que resulten inadecuados o excesivos, pero solo existen tres campos para los usuarios, email nombre de usuario y contraseña, pero ninguno se usa para nada más que autenticarse. Por último, el derecho de oposición, por el que una persona puede optar por el cese del uso de datos personales que no haya consentido usar, o que sean usados con finalidad publicitaria. De manera similar al derecho de cancelación, no es aplicable en este caso, ya que no se hace uso de la información para usos publicitarios, y siempre son de *consentimiento implícito*. Por último, existe un derecho excepcional, el derecho al olvido, que contempla la eliminación de registros de acceso público, que aun pudiendo ser verídicos y legítimos, el usuario quiere que no sean de acceso público. En este caso, si bien no existen datos de acceso público, cuando un usuario es borrado del sistema, es borrado completamente, no se deja en el sistema como inactivo u otras prácticas realizadas comunmente en el entorno profesional.

Por último, la RGPD contempla la necesidad de determinar un encargado de Protección de Datos, si los datos a manejar son de nivel medio/alto, lo cual no es el caso en este sistema, por lo que no es necesario dicho encargado.

6.2.2. Derechos de autor

Este sistema permite subir registros con imágenes de documentos manuscritos, o escritos a máquina. Esto puede suponer problemas de derechos de autor, si las imágenes pertenecen a una obra protegida por derechos de autor (Art. 12 de la Ley 23 de 1982). Por lo que si las imágenes de un registro (ya sean a mano o no) pertenece a una obra protegida, se estaría infringiendo los derechos de autor.

No se ha contemplado durante el desarrollo del proyecto, pero sería necesario indicar o bien a la hora de crear un registro (y añadir imágenes a un registro), o en un hipotético tutorial por el que los administradores tendrían que pasar, que se deben revisar los derechos de autor, a la par que añadir un *disclaimer* informando de que la plataforma (ni su creador) tienen ninguna responsabilidad legal sobre las acciones que sus usuarios lleven a cabo.

6.3. Contexto socioeconómico

Como se ha mencionado previamente, a priori hay tres escenarios muy claros en los que una herramienta de este tipo podría ser usada, y un uso (no tan claro) para el sector privado:

- Digitalización de registros eclesiásticos. Ayudaría a mantener un sistema informatizado de todos los registros creados hace más de 25/30 años, a la par que abriría las puertas a otras posibilidades, por ejemplo, poder trazar tu árbol genealógico de forma muy sencilla
- Recurso adicional para bibliotecas que alivia el trabajo manual, y añade la funcionalidad extra de poder buscar por texto en las imágenes. Actualmente, como se ha mencionado previamente en este trabajo, las bibliotecas manejan manuscritos, pero solo permiten búsquedas por título y autor, y no permiten visualizar el texto. Con esta herramienta la forma de buscar amplía, y sí se puede visualizar el texto
- Ayuda a asociaciones de estudiantes, o estudiantes en general sin necesidad de asociación, que permite compartir apuntes de forma más dinámica, sencilla, y accesible.
- En el sector privado, en cualquier empresa siempre hay multitud de documentos internos que establecen una serie de reglas, como pueden ser documentos de seguridad, protocolos de evacuación, protocolos de comunicación, de facturación... Un posible caso para una empresa privada

podría ser usar esta herramienta (o la parte de búsqueda en documentos por texto) para permitir a los empleados buscar un término o regla en concreto.

Debido a que casi todos estos ámbitos no son de carácter típicamente comercial, la forma de explotar económicamente un producto como este tiene una nivel de complejidad añadido, contando además con el hecho de que son organismos públicos. Se podría establecer un sistema de pago anual, que corresponde con el número de creación de registros de la plataforma, y el número de búsquedas. Sería por lo tanto un servicio cuyo precio es basado en la demanda.

7. Conclusiones y trabajo futuro

7.1. Conclusiones

El propósito principal del trabajo era crear una interfaz que permitiese la búsqueda en documentos manuscritos por texto. Dicho objetivo ha sido alcanzado, a pesar de las posibles mejoras a la usabilidad del producto que se han detectado en la sección de experimentación.

Hasta donde sabemos, esta herramienta propone varias mejoras respecto a las existentes. Por un lado, permite gestionar de forma sencilla los documentos tanto manuscritos como no, y traducirlos a texto plano sobre el que se pueden hacer búsquedas de forma sencilla. En cuanto a la traducción de imágenes a texto, el Machine Learning se ha postulado como un conjunto de técnicas que permiten llevar a cabo esta tarea de forma muy correcta. Pero si no fuera así, la herramienta permite la posterior edición del texto traducido por parte del administrador. Además, la herramienta permite identificar claramente cuáles son las hojas del documento que contienen la palabra o frase buscada.

Por último, la realización de un proyecto como éste, en el cual he llevado a cabo desde el diseño, hasta la validación final, pasando por implementación, e investigación necearia para estudiar viabilidad del sistema y uso de tecnologías, ha proporcionado una experiencia y aprendizaje inestimables, que sin duda podrán ser usados en mi trayectoria profesional.

7.2. Trabajo futuro

Esta plataforma puede dar mucho de sí, hay muchas más funcionalidades que se pueden implementar. Como trabajo futuro, se propone:

- Dado el resultado en las entrevistas de usuario, que muestran algunos aspectos de la interfaz que podrían ser mejorados, o hecho de forma diferente, la primera mejora que se propone es hacer un cambio en la interfaz, y realizar tests A/B, con la interfaz antigua y la nueva, para ver cuál funciona mejor. Adicionalmente, si se implementa otra de las mejoras propuestas más abajo, un tutorial de la plataforma, habría que hacer el estudio con un totla de cuatro grupos, nueva plataforma sin tutorial, nueva plataforma con tutorial, antigua plataforma sin tutorial, y nueva plataforma con tutorial.

- Buzón de sugerencias, para los casos de iglesias y asociaciones de estudiantes que recogen apuntes, tener un buzón de sugerencias, en el que un usuario no registrado puede subir imágenes, con un título, a modo de sugerencia. Por ejemplo, un estudiante de ingeniería que acabe de terminar segundo de carrera, y tiene sus apuntes de Cálculo III escaneados, puede subir esos apuntes escaneados a la plataforma de una asociación para que más gente pueda disfrutar de ello.

- Creación de roles de usuarios, con varios niveles. El sistema tendría un pequeño elenco de administradores, con permisos para todo, y por debajo habría dos tipos de usuarios adicionales: subadministradores, los cuales tendrían permisos parecidos a administradores, pero no tienen permitido borrar nada, y los *curators*, o revisores, cuyo trabajo se centraría en editar contenido, revisar que el texto de las imágenes sea el correcto, y revisar las sugerencias del buzón de sugerencias, para comprobar si vale la pena subir el contenido.

- Sería lícito estudiar la posibilidad de implementar un sistema de corrección de texto que se encargue de revisar el texto que el algoritmo de ML proporciona, para arreglar pequeños fallos ortográficos derivados de incorrecto reconocimiento. Existen técnicas de NLP [Gupta and Mathur, 2012] que sirven para este propósito

- Debido a que estaba fuera de la intención del proyecto el sistema de contraseñas y modificación de las mismas se ha mantenido lo más sencillo posible, pero en un sistema real, es necesario implementar un sistema de emailing para este propósito. Cuando un usuario necesite cambiar su contraseña, en la página del login tendría la opción de ”recordar contraseña”, tras lo cual recibe un correo, o un usuario administrador (o subadministrador, en caso de implementar la jerarquía de roles), puede activar la opción de mandar el correo a otro usuario para que pueda cambiar su contraseña.

- Algo que puede ser necesario llegado un cierto nivel de utilización, es la necesidad por parte de un administrador, de modificar el orden en el que las imágenes están guardadas. Esto supondría varios cambios, el primero, en el cliente, habría que modificar la interfaz de edición de imágenes, y la de añadir, para poder reflejar la posibilidad de añadir, o mover, en una determinada posición una imagen. Y en segundo lugar, en el back, habría que modificar el modelo Imagen, añadiendo el campo index, que reflería a qué número dentro

del registro corresponde la imagen, y las correspondientes vistas que manejan la edición de la imagen.

- Dependiendo del caso y uso, los modelos de la base de datos (y por tanto, las vistas en front, y las vistas en Django) podrían ser modificados. Por ejemplo, como se menciona en la Sección 5.1, la vista detallada inicialmente estaba pensada para que en un futuro tuviese más información, como por ejemplo el resumen del documento. En el caso de apuntes de estudiantes, podría tener un campo para Asignatura, otro para Curso y otro para Carrera.

- Como se puede ver en las entrevistas a usuarios, en algunos casos, debido al desconocimiento previo, encontrar alguna funcionalidad en concreto les era difícil. Por esto, una mejora futura sería realizar un pequeño tutorial, ya sea un video corto, o en la propia interfaz, a modo explicativo para dar a conocer todas las funcionalidades.

- Esta aplicación se mueve en un delicado sistema legal de copyright, sería interesante añadir un apartado (en el footer de la página) que contenga la información legal, y qué responsabilidad contiene cada parte. Adicionalmente, y a medida de seguridad técnica y legal, se debería crear un sistema de auditoría, en el cual se recoge quién ha realizado determinadas acciones (se haría solo con aquellas que pueden resultar dañinas o ilegales, como son crear un registro, y añadir imágenes a un registro)

- Conjuntamente con el tutorial, o de forma independiente, una sección de Frequently Asked Questions (FAQ)

- El cliente no se ha hecho de forma que sea compatible (totalmente) con un móvil, por lo que abrir la aplicación en un dispositivo móvil o tablet resultará en una interfaz inconexa o confusa. Adaptar el código del cliente para hacerlo mobile-friendly sería una posibilidad a estudiar.

- Similarmente al caso anterior, yo carezco de conocimiento para hacer una interfaz que sea completamente accesible para todo tipo de discapacidades, por lo que un estudio sobre cómo mejorar la accesibilidad (y posterior implementación) sería una mejora interesante.

- Añadir idioma al cliente, permitir la posibilidad de tener la interfaz en un idioma diferente al español.

- Añadir idioma al servidor, a la hora de realizar la búsqueda, añadir un filtro de búsqueda en el que se pueda seleccionar el idioma del texto a buscar.

- Una forma de mejorar la usabilidad (se hace una mención en la parte de testing a usuarios) es añadir *loaders* en la página. Los loaders son pequeñas indicaciones visuales de que algo está pasando, y se espera una respuesta u otro estímulo (por lo general por parte del servidor al que han hecho una petición).

Anexo A Documentation

El código tanto del servidor web, como del servidor front, como el usado para comparación de técnicas ML se puede encontrar en el repositorio

https://github.com/ForFer/bachelor-thesis

Anexo B Summary in English

B.1 Introduction

B.1.1 Problem Description

In an ever more digitalized world, it is still possible to find handwritten documents, in places such as libraries, churches, or hand notes from students. However, efforts by libraries to digitalize such documents is on the rise, in order to make them accessible to everyone. Examples of such efforts are Biblioteca Nacional de España [BNE, 2019] and Prado Museum [MP, 2019].

While searching through these documents is possible by author, year, title . . . it is rare to find search by text within these documents. It is a waste of time to go page by page through a document in order to find something specific, it would be much better to offer this functionality.

Currently, there are several techniques and tools to analyse machine-like fonts, one of the most used is Optical Character Recognition (OCR), currently based on Machine Learning techniques [Verma and Ali, 2012], but it is still not widespread on its use for handwritten recognition. Additionally, transcribing images must be done by hand, one by one.

In this thesis a solution to this problem is proposed, by presenting a tool that allows managing documents, with automatic handwritten text recognition from images, and allows to search by text within those documents.

B.1.2 Objectives

- Requirements specification. Since the initial description of the thesis is purposely vague, it is necessary to gather and detail all the requirements.

- Web technologies analysis. Web development is a very changing domain, with varying paradigms throughout the years. Currently it seems like Client Server Rendering (CSR) is a better solution for certain types of applications.
- Machine Learning (ML) techniques analysis. Since there are more than one way of recognizing text, techniques should be analyzed to pick the best (or fittest to our problem)
- Design of the system. Once analysis of necessary technologies (web and ML) has been done, and requirements have been gathered, a proper design is due. For our tool, a decoupling of back and front will be done, having Python Django for the back, acting as API, and Node React for the front, acting as Single Page Application.
- Implementation. After the design is finished, the tool will be implemented. This tool will allow managing documents, creating new entries, modify them, and eliminate them. It will be possible to perform search by text, plus filter by author, title, and year. Once search has been done, results will be shown to the user.
- Experimentation. ML techniques will be tested to analyze which one performs better, and usability (UX) will be tested.
- Planning. To carry our the project, previous planning is needed, budget and time constrains will be taken into account. Also, socioeconomic impact, and the legal domain of the tool.

B.1.3 Structure of the document

1. State of the art
2. Description of the system
3. Implementation
4. Testing
5. Organization of the project
6. Conclusions and future work
7. Annex

B.2 State of the art

Before going into detail with design and implementation, it's important to detail the state of the art to understand the context under which this thesis has been made. Starting with web paradigms, to continue with handwritten text recognition techniques used throughout time, mostly those related to Machine Learning

B.2.1 Web paradigm

Web applications have typically used Model View Controller (MVC), where *model* refers to the data handling, the *view* refers to the part that shows information to the user, and finally, the *controller*, handles user input, which in turn handles the model. Figure 1 shows the interaction between these three elements. Recently, a new paradigm is being used more and more, as can be observed at figure 2 (Google Trend results since 2004 until today). This new paradigm is based on decoupling the service that handle visualization of data, and the service that handles user input and data handling [Marcio Galli and Oeschger, 2003]. This means, having an API, which handles user input and data, and a Single Page Application (SPA) which shows the data.

A website has three elements, HTML, which holds the structure of the website; CSS, which beautifies the structure; and JS, which is used for the user-site interactions, and modification of HTML. Typically, with SSR whenever any interaction is made, the whole website had to refresh, since the rendering is done in the server, but with CSR, since the rendering is made in the client, there is such need, all the code is sent once, and only data is reloaded. Python and React has been used for this purpose, with PostgreSQL as database.

B.2.2 Python

Python is a multiparadigm programming language, characteristic for its dynamic type and being an interpreted language. This means Object Oriented (OO) and functional programming is supported in Python, variable types do not need to be specified, they will be inferred during execution, and there is no need for compilation. Due to these characteristics, it's easy to use and learn, the use is very widespread, ranging from educational purposes to ML, scientific computation... Makes the language very flexible, but it is also slow (from being interpreted), there are

currently two major versions (Python2, and Python3) supported (Python2 until 2020), and soon Python4 will make its debut, and it is not very friendly to debug (again, from being interpreted). The advantages overweight the disadvantages, that is why Python was chosen for the project.

There are several Python frameworks to develop a website. Some of the most famous, and supported are Django, Flask, and CherryPie. Django was chosen for the project since within the Fullstack (those who provide templating and data handling) frameworks was one of the best. There are many built-in functionalities that come with the default Django implementation, and many more modules that are easy to install to use alongside Django.

B.2.3 React

Within the many frameworks to use SPAs, some of the most common ones are AngularJS, React, and VueJS. Due to my lack of knowledge, and a bigger community than the others, React was the chosen front framework for the project.

React was created by Facebook, with the sole purpose of showing data, unlike its competitors, which makes it lighter. This JS framework provides a templating system that renders HTML, while keeping an internal state per component. This state can be used to keep track of form fields, of whether an element was clicked...

This framework is based on Components, mentioned previously. A component is the basic unit which is used to render. A component can call other components, creating a hierarchy, and can be dumb, holds no state, it exists purely to show data; or smart, which holds data (and it's more likely to be used to handle interactions with the user). Code samples 1 and 2 show a dumb and a smart component respectively.

B.2.4 PostgreSQL

Django comes with hooks to four databases by default: Oracle, SQLite, MySQL, and PostgreSQL. Out of those four databases, all of the relational type, only one has Full Text Search built-in, PostgreSQL. This technology provides a superior search engine when it comes to text, since it allows for many different types of search within text, which would have to be manually implemented in the other databases.

B.2.5 Machine Learning

Neural Networks. NN is mankind attempt to imitate brain cells in a computer. They are created with the purpose to find patterns, learning from examples, and generalize data in the best way possible. The origen of this technique dates back to 1943 [McCulloch and Pitts, 1943], where a Perceptron was first defined, a NN with only one neuron, and one layer. Figure 5 is a visual represention of said neuron. Each neuron in a NN is composed of three parts, the inputs $(x_1, ..., x_N, andbias)$, each with a weight $(w_1, ..., w_N)$, the neuron itself that has an activation function, and the output, which is the result of the activation function. The output of a neuron comes from the result:

$$y = w_1x_1 + ... + w_Mx_N + w_b = \sum_{\mathtt{i}=1}^{N} w_{\mathtt{i}}x_{\mathtt{i}} + w_b$$

But, how does a network of neurons with this function detect patterns? By training the network with examples. The network is initialized with random weights, and examples are passed to the network, through each neuron. The result of each iteration is analyzed, and depending on whether the approximation is correct (based on the Error calculation), the weight keeps changing towards a certain *direction*, determined by the Gradient descent (represented in figure 6).

There are several types of NN, some of them are: MultiLayer Perceptron, which is a NN with several layers, each of them has all their neurons connected to all the neurons in the next layer; Convolutional NN, characterized by having convolution layers (a convolution layer is specialized in detecting visual patterns); and Recurrent NN, in which there is an internal state at each neuron that acts as memory, and there is an extra input value to make the neuron forget the previous value.

Handwritten text recognition (HTR)

There are two versions of this problem, offline and online. Online HTR implies real time recognition, for instance, in a tablet, or phone. Offline version only analyzes an image to try to extract text from it. Online HTR has been pretty much solved, and is generally accepted that offline is harder, since there is less information to work with (in online HTR, the current trace direction and speed are analyzed, together with past information).

Offline HTR is divided in subtasks to be done properly:

1. Find text in the image, and segment the text
2. Segment each line individually (which is harder when handwritten text is present)
3. Segment each line into words
4. Recognize the word. This last part has been tackled in two different ways, either by trying to recognize the whole word as a unit, or by segmenting it further and then trying to find characters.

Throughout time, several approaches have had different success, from using several classifiers for word recognition, to using Markov Models. But it is in 2009 when there are two breakthroughs, first one [Plötz and Fink, 2009] uses a model with Hidden Markov models and Markov Chains embedded, and is able to realize all the steps aforementioned. They use as a metric to measure the error Word Error Rate (WER), in which they analyze Substutions, Deletions and Insertions against Total number of words. In general datasets they have a WER of 10 % to 40 %, but on specific datasets, where the model is better trained, the WER lowers to 0.8 % to 12 %.

The second big breakthrough comes thanks to Graves [Graves and Schmidhuber, 2009], which is based on three components:

- MultiDimensional Recurrent NN (MDRNN). One dimension to traverses an image in each of the possible directions from the 4 corners, from top-left to bottom-right; from bottom-left to top-right; and so on. This network will feed another layer, that is also traversing the image, with the context at each point.
- Connectionist Temporal Classification. Output layer in the network that labels RNN sequences. This layer has one more unit than elements of the alphabet $\mathcal{L}$, to predict the labels, blank space, o no label.
- Network hierarchy. The last "component", is the way to structure the network hierarchy. Given an image, divide it in blocks, send each block to the MDRNN, store the activations in blocks, and feed this information to the next layer.

The results from these layer are quite good. The paper is made by non-arabic speakers, which were able to train the network to understand handwritten arabic

text, with an accuracy of 91.43 % to 96.75 % (in two competition modalities).

These models are not accessible to the general public, and training a model from scratch is quite costly (time-wise and money-wise). That is why APIs that provide this functionality are a good alternative; two of them are Google Vision API [Google, 2019], which does not reveal what techniques are used, but it is safe to assume a combination of previously mentioned is the most logical. The second API is Microsoft Azure Vision [Microsoft, 2018], which works with Faster R-CNN (a combination of recurrent and convolutional NN).

B.2.6 Digital Libraries

Typically the task of transcribing handwritten texts to a database has been done by libraries. and student associations. Some examples are [MP, 2019, BNE, 2019, BiblioFCC, 2019]. For instance, Biblioteca Nacional (figure 8) shows that it is possible to search by author, title, date... but **not** by text. Very few allow to search by text, and the few that do, there is no indication of which pages contain the result.

Google Books is another of the few resources that allow to search by text within a document, but it is rare to be able to access the whole document. Furthermore, as far as I am aware, it does not contain handwritten documents.

B.3 System Design

B.3.1 System Architecture

As previously mentioned, this project follows a decoupled paradigm. The full architecture is shown in image 42, where the different components and its connections are shown. The front only connects to the client, and the client only makes requests to the back. In turn, the back, where security is handled, ”talks” to Google Vision API and the Database server.

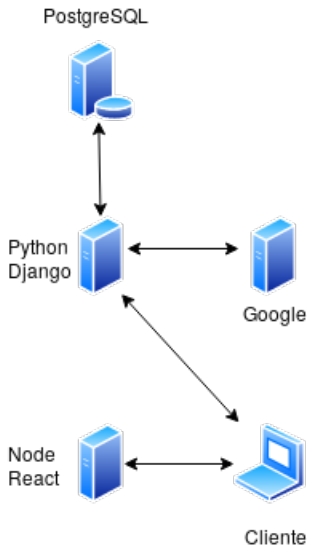

Figura 42: System Architecture

B.3.2 Back - Django Rest Framework

Since we are using a decoupled approach, Django-Rest-Framework (DRF) is used. This framework allows us to create a REST (Representational State Transfer) API, which will receive the requests from the client.
Django (with, or without DRF) has an implicit development ”flux”:

- First, models are defined. A model is the Django representation of a database table. For instance, code sample 21, which is Python code, represents what is shown in code sample 22, which is the creation of a Table in SQL.

Código 21: Django Person model

```
from django.db import models

class Person(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
```

Código 22: SQL Person Table

```
CREATE TABLE myapp_person (
    "id" serial NOT NULL PRIMARY KEY,
    "first_name" varchar(30) NOT NULL,
    "last_name" varchar(30) NOT NULL
);
```

One of the main advantages of Django is that with what is shown in code sample 21, all the CRUD (Create Read Update Delete) operations are covered automatically.

- Second, Serializer. Models are high-level data structures that Django creates in order to facilitate handling data. In order to handle requests and send responses with that data, a "translation" to a simpler data structure is needed. Django provides a way to do this with Serializers.
- Third, View. While Django (somewhat) follows MVC paradigm, a view in Django is not the same view as MVC. In this framework a view is the logic, where data modification is handled. It would be equivalent to the controller. To modify the data and serve requests, the Serializers are used.
- Lastly, routing. It is necessary to tell Django which endpoints (URLs) should be listened to, and for each endpoint, a view should be provided.

Models

Models have been created following the schema shown in picture 43

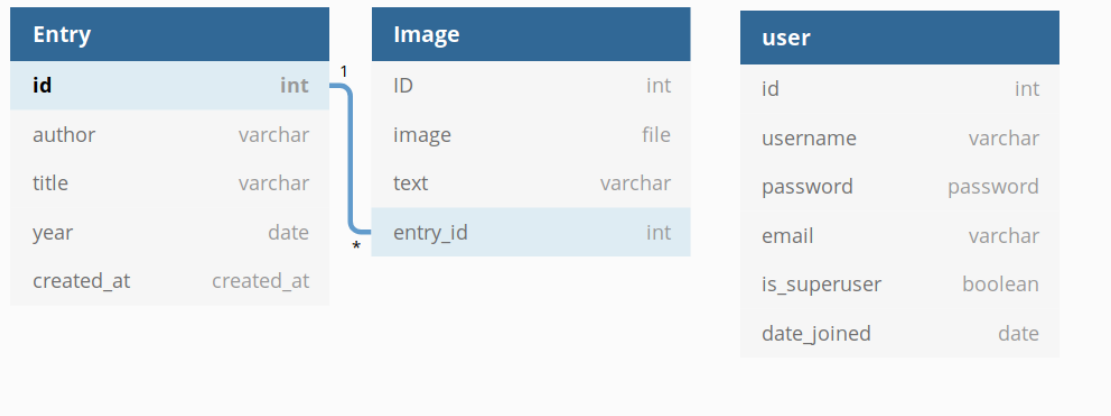

Figura 43: Database model

Security

There are several threats that need to be taken into account when developing a web server.

- Cross-Origin Resource Sharing (CORS). A security measure to allow requests only from URLs from a whitelist that is kept in the server configuration site.
- Cross-Site Request Forgery (CSRF). A threat that is based on using the cookies of a browser to make seemingly harmless requests, once the user has been authenticated somewhere.
- Cross-Site Scripting (XSS). An attack that allows the attacker to inject code in a website source code, and would give the attacker control of credentials, and to impersonate the victim by making requests on their behalf.
- SQLinjection. This attack works by injecting SQL code in the database, which is then executed, when it should not.
- Others. Some other key factors are: deploying over HTTPS, so that content is encrypted in the channel that is sent, never allowing *GET* requests to modify data, only read, handle very carefully sensitive information, such as passwords (never store in plain text, or send passwords from other users)

B.3.3 Front - React

There are four views in the web application, Landing, Login, Search and Profile, plus a Header, which is not a view in itself, rather a component that all the views, except Login, use. They are connected as shows in figure 44.

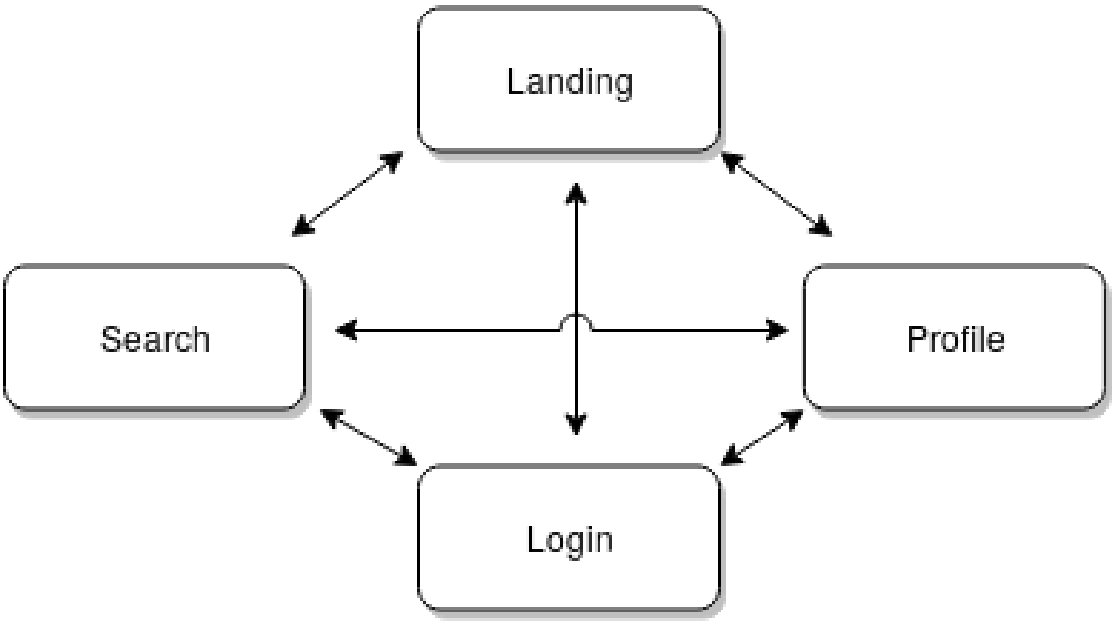

Figura 44: View Diagram

- Header. It has two purposes, first, act as standard throughout the site, at any view you are, you will see a header, that is the same, this helps towards user experience. Second, it handles redirections: if a user wants to log in, or is logged in and wants to go to profile, or wants to head back home. All of these actions are handled by the Header component.
- Landing. First view that is shown to users, with a very simple UI, the only elements in this page are, the logo at the top left, Login top right, and a search bar. All aimed to the general public (since it is expected that more than 80/90 % of the users will be non-registered users). Figure 16 shows how it looks like.
- Login. Following Landing simplicity, this view literally has only one form to log in, and an arrow to go back to where you were before. Figure 17 shows how it looks like.
- Search. This page contains the search results that are shown after performing a search in the Landing view. Striving for simplicity and easy use, the information displayed to the user is divided in: header on top; and below it, in two columns, the filters and order criteria on the left (around 15 % of the width of the screen) and the search results on the right, with a search bar. Figure 18 shows how it looks like.
- Profile. Finally, the most ”advanced” view. After loging in, a user will be redirected to this view. It is divided in three parts, header on top;

below it the category and subcategory picker, and below the content to be shown, depending on the category and subcategory selected. This view, only accessible to registered users, allows such users to query, add, delete, or edit entries (including the modification of images per entry, adding, removing of images, and modifying the text that the ML algorithm recognized automatically), and also allows to manage users. Figure 22 shows how the general profile looks like.

B.3.4 Machine Learning

As has been mentioned before, Google Vision API will be used for this project. In order to use it, one must follow steps described in images 34 and 35. Following those steps will yield in having a Google Cloud Platform account, with a Service account, having a key in a *.json* file. With that key, after exporting as shown in code sample 18, it is possible to start using already the API.

In order to implement this API in the web server, code sample 23 shows what was introduced in file *settings.py*. *text_ detection* module is shown in code 19. This module is called when the model Image is saved (but not edited by a user)

Código 23: Vision API in *settings.py*

```
from text_detection.detector import detect_document
DETECTOR = detect_document
```

B.4 Testing

There are two main issues to test in this project. The first, user interviews, serve the purpose of checking the usability of the product with real world users, of different background. Secondly, providing a testing mechanism for ML techniques, by benchmarking both APIs, but the mechanism can be used for future techniques or algorithms in the same way.

B.4.1 User interviews

There were two phases in the user interviews. An early phase, when product was not finished 100 %, which helped find early usability issues, and other bugs.

The second phase is structured in two parts: during the first one, the interviewees were asked to perform several tasks, with the bare minimum explanation of the tool, in order to see how intuitive the system is without the need of a user manual. Their behaviour and observations were noted down; the second one was quantitative, they were asked to rate from 1 to 5 several aspects of the process and the tool itself.

The results were overall positive, with some exceptions in certain interactions. For instance, there was a detailed view of the entry which no one liked. The task to edit text in a certain image, after a search query was particularly tricky, since many were not able to find too fast the option to edit it, nor did all of them realize that there were several images to edit, rather than only one. On the quantitative part, the results were again positive, most of them rated rather high whether they would like to use the system, and if they found the system easy to use. On the other hand, on the inconsistency and need for technical support they rated low (as in, they would not need support, and didn't find inconsistencies).

B.4.2 Machine Learning

Due to the nature of the task, in order to compare the performance of the different models, text similarity metrics have been used. There are three types [Gomaa and Fahmy, 2013] and only the type **String-base** will be used. From that type, a total of four different techniques are going to be used:

- Damerau-Levenshtein. Difference based on the minimum number of operations needed to transform one string into another.

- Damerau-Levenshtein Normalized. The normalized version of the previous technique, so it measures the relative Damerau-Levenshtein distance.
- Jaccard Similarity. Number of words shared between two strings, only taking into account unique ocurrences of words (disregarding frequency)
- Cosine similarity. A vector composed of frequency of *sufficiently relevant* words is created, and the similarity is the angle projected by the intersection of the two vectors

For this test, five different images have been used, four handwritten in different formats, sizes, and noise levels (images 36, 37, 38, 39 and 40). The results are shown in tables 51, 52, and 53.

Jaccard Similarity					
	36	37	38	39	40
Google	0.0	0.0	0.546	0.425	0.889
Azure	0.0	NA	0.427	0.31	0.7

Cuadro 51: Jaccard Similarity results

Cosine Similarity					
	36	37	38	39	40
Google	0.0	0.0	1.981	1.793	1.789
Azure	0.0	NA	1.781	1.559	1.604

Cuadro 52: Cosine Similarity results

Damerau-Levenshtein					
	36	37	38	39	40
Google	3	42	60	89	5
Google (normalizada)	0.0385	0.857	0.103	0.130	0.125
Azure	6	NA	42	106	4
Azure (Normalizada)	0.0759	NA	0.0725	0.156	0.108

Cuadro 53: Damerau-Levenshtein results

Jaccard Similarity (from 0 to 1, the higher the more similar the strings are) shows that Google API perform better in each of the images. Images 36 and 37 do not have a value since words with length 1 are not taken into consideration. The same happens with Cosine similarity (values 0 to 2, the higher the more similar),

where again, Google API outperforms Azure API in every image. Finally, looking at the Damerau-Levenshtein distance (and it's normalized version), the results here are not as favourable to Google as the previous one, since images 38 and 40 have a higher distance (by 3 % and 2 % respectively), but on the rest it still gets better results than Azure. Furthermore, image 37, which is the artificially created noisy image, cannot be processed by Azure, whereas Google, albeit ostensibly bad, gets a result.

Since Google API has proved to perform better in most of the experiments, it was the chosen one for the project. Additionally, Azure API creates an overhead in the text recognition task, by providing a URL after making a request to recognize, to which a second request must be done (and there is some delay to get the results from the second URL).

B.5 Project Organization

B.5.1 Planning

This project is divided in several phases:

- Analysis. Requisite specification is made from the problem statement.
- Research. State of the art is researched, evaluating the possible technologies in the process.
- Design. The design of the architecture, different parts of the system, how will the servers and client communicate...
- Implementation. Development of the system
- Testing. User interviews to test the UI and UX.
- Documentation. Final phase to note down all the results obtained from the previous phases.

Figure 41 shows the Gantt diagram of the different phases. As can be observed, design e implementation went hand in hand during the most part of the duration of these phases, and that is because SCRUM methodology has been used. SCRUM is an agile methodolody that is based on short sprints, each of which has several user stories to develop. Also, it has the purpose to create a Minimum Viable Product (MVP) as soon as possible. In this case, the MVP was reached soon, but for each sprint, new design of endpoints, and how to pass data to and from that endpoint was needed.

Besides time planning, it is also necessary to propose a budget for the tool, given the expected development time, tools used, and staff that should be hired. The final cost is shown in table 54

Concept	Total
Staff cost	8350.00 €
Equipment cost	66.67 €
VAT (21 %)	1767.49 €
Project Costs	**10184.16€**

Cuadro 54: Total project cost

Additionally to this budged (which would be just the needed amount to develop the project itself), if this application were to be commercialized, it would need to have a price per usage, since that is the pricing model of Google API.

B.6 Conclusions and Future work

B.6.1 Conclusions

The main purpose of this thesis was building a tool that allows search by text in handwritten documents. Said objective has been fulfilled, despite the possible UI/UX improvements.

As far as we know, this tool improves existing solutions. On one hand, it provides an easy way to manage documents, and transcribes the images to text. On the other hand, while Machine Learning techniques have been used to the automatic transcribing of image to text, administrators can manually correct the possible mistakes in the text. Furthermore, the tool indicates clearly which pages of the document contain the search results.

Lastly, on a personal note, carrying out a project like this, from design to final validation, going through research and implementation, has provided me with an invaluable experience and learning process, which without a doubt will be of use in my career.

B.6.2 Future Work

This tool has potential to be very useful in many different domains, if some of these functionalities or suggestions are followed. As future work it is proposed:

- Given the usability interviews result, it looks like a small redesign in UI/UX is due, unless a user manual is given. (Users got used to the category and subcategory picker in the Profile view rather quickly, but not from the beginning)
- Suggestion box, to allow non-registered users request a document, or for them to upload a document. For instance, in the student notes case, students can upload their own personal notes, and a curator role would oversee the process.
- More roles, as mentioned in the previous item, if the suggestion box is implemented, a *curator* would be necessary, which only would have the option to create content, or edit text in an image. Additionally, a sub-administrator role, without deletion permissions would be necessary.
- There are currently NLP techniques that can correct spelling mistakes [Gupta and Mathur, 2012]. Combining such system with the ML text recognition is a possibility to study.

- A In a production level system, the passwords are handled in a different way, with a mailing system instead of being modified by other users. An emailing system that allows user to confirm their account, once they are registered, and to change their password if forgotten would be necessary.
- It may necessary, given a sufficient volume of documents, or size of each document, to be able to modify the order of the images within the document.
- Models in the Database may need modification per use. For instance, Entry may need an *abstract* field, or for the students' notes, *subject*, and *degree*.
- A tutorial or user manual would be of great help, as seen in the usability interviews.
- Currently the front is not completely mobile friendly, it is a necessary step to be taken if this system wants to be used
- Study the accessibility in the website is also a good future step to take, and try to improve the accessibility as much as possible
- The tool is currently in Spanish, with no option to change it. At least English would be the minimum (even if the tool is used only inside Spain)
- Another way to improve usability is implementing *loaders* in the front whenever a request to the web server has been made. It'll help towards informing the user an action is happening.

Referencias

[AEPD, 2018] AEPD (2018). Guía del reglamento general de protección de datos para responsables de tratamiento. https://www.aepd.es/media/guias/guia-rgpd-para-responsables-de-tratamiento.pdf.

[Banko and Brill, 2001] Banko, M. and Brill, E. (2001). Scaling to very very large corpora for natural language disambiguation. In Proceedings of the 39th Annual Meeting on Association for Computational Linguistics, pages 26–33. Association for Computational Linguistics.

[BC, 2019] BC (2019). Manuscritos. https://www.bnc.cat/esl/Fondos-y-colecciones/Manuscritos.

[BiblioFCC, 2019] BiblioFCC (2019). Apuntes digitalizados. http://bibliofcc.com.ar/bibliofcc/.

[BNE, 2019] BNE (2019). Catálogo de manuscritos de la biblioteca nacional de españa. http://catalogo.bne.es/colecciones/manuscritos/colManForm.htm.

[BOE, 2018] BOE (2018). Boletín oficial del estado. https://boe.es/boe/dias/2018/12/06/pdfs/BOE-A-2018-16673.pdf.

[CSIC, 2017] CSIC (2017). Fondo de archivo del instituto de física fundamental. http://www.cfmac.csic.es/biblio/FondoIFF.html.

[Doetsch et al., 2014] Doetsch, P., Kozielski, M., and Ney, H. (2014). Fast and robust training of recurrent neural networks for offline handwriting recognition. In 2014 14th International Conference on Frontiers in Handwriting Recognition, pages 279–284.

[FastAI, 2019] FastAI (2019). Learning rate finder. https://docs.fast.ai/callbacks.lr_finder.html.

[Fletcher, 2019] Fletcher, N. (2019). Classification vs detection vs segmentation models: The differences between them and when to use each. https://blog.clarifai.com/classification-vs-detection-vs-segmentation-models-the-differences-between-them-and-how-each-impact-your-results.

[Gomaa and Fahmy, 2013] Gomaa, W. H. and Fahmy, A. A. (2013). A survey of text similarity approaches. International Journal of Computer Applications, 68(13):13–18.

[Google, 2019] Google (2019). Google vision api. https://cloud.google.com/vision/.

[Graves and Schmidhuber, 2009] Graves, A. and Schmidhuber, J. (2009). Offline handwriting recognition with multidimensional recurrent neural networks. In Advances in Neural Information Processing Systems 21, pages 545–552. Curran Associates, Inc.

[Gupta and Mathur, 2012] Gupta, N. and Mathur, P. (2012). Spell checking techniques in nlp: a survey. International Journal of Advanced Research in Computer Science and Software Engineering, 2(12).

[Géron, 2017] Géron, A. (2017). Hands-On Machine Learning with Scikit-Learn and TensorFlow. O'Reilly Media.

[Hosseini et al., 2017] Hosseini, H., Xiao, B., and Poovendran, R. (2017). Google's cloud vision api is not robust to noise. In 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA), pages 101–105.

[Impedovo et al., 1991] Impedovo, S., Ottaviano, L., and Occhinegro, S. (1991). Optical character recognition - a survey. International Journal of Pattern Recognition and Artificial Intelligence, 05(01n02):1–24.

[Jaworski and Ziadé, 2016] Jaworski, M. and Ziadé, T. (2016). Expert Python Programming (2nd ed.). Packt Publishing.

[Kaliski, 2000] Kaliski, B. (2000). Pkcs 5: Password-based cryptography specification version 2.0.

[La-Bisagra, 2018] La-Bisagra, A. L. (2018). Apuntes digitalizados. https://asociacionlibrepsico.wordpress.com/2018/04/18/apuntes-digitalizados-2018/.

[Marcio Galli and Oeschger, 2003] Marcio Galli, R. S. and Oeschger, I. (2003). Inner-browsing: extending the browser navigation paradigm. https://developer.mozilla.org/en-US/docs/Archive/Inner-browsing_extending_the_browser_navigation_paradigm.

[Marti and Bunke, 2002] Marti, U.-V. and Bunke, H. (2002). The iam-database: An english sentence database for offline handwriting recognition. International Journal on Document Analysis and Recognition, 5:39–46.

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics, 5.

[Microsoft, 2018] Microsoft (2018). Classification vs detection vs segmentation models: The differences between them and when to use each. https://www.microsoft.com/developerblog/2018/05/07/handwriting-detection-and-recognition-in-scanned-documents-using-azure-ml-package-computer-vision-azure-cognitive-services-ocr/.

[Mozilla, 2019] Mozilla (2019). Cross-origin resource sharing (cors) - http. https://www.postgresql.org/about/.

[MP, 2019] MP (2019). Buscador biblioteca digital. https://www.museodelprado.es/aprende/biblioteca/biblioteca-digital/buscador.

[Pan and Yang, 2010] Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. IEEE Transactions on Knowledge and Data Engineering, 22(10):1345–1359.

[Plamondon and Srihari, 2000] Plamondon, R. and Srihari, S. N. (2000). Online and offline handwriting recognition: A comprehensive survey. In IEEE Transactions on Pattern Analysis and Machine Intelligence (vol 22, no. 1), pages 63 – 84. IEEE.

[Plötz and Fink, 2009] Plötz, T. and Fink, G. A. (2009). Markov models for offline handwriting recognition: a survey. International Journal on Document Analysis and Recognition (IJDAR), 12(4):269 – 298.

[PostgreSQL, 2019a] PostgreSQL (2019a). About postgresql. https://www.postgresql.org/about/.

[PostgreSQL, 2019b] PostgreSQL (2019b). Chapter 12. full text search. https://www.postgresql.org/docs/current/textsearch-intro.html#TEXTSEARCH-DOCUMENT.

[Rahul Kala and Tiwari, 2010] Rahul Kala, Harsh Vazirani, A. S. and Tiwari, R. (2010). Offline handwriting recognition using genetic algorithm. CoRR, abs/1004.3257.

[Ren et al., 2015] Ren, S., He, K., Girshick, R. B., and Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. CoRR, abs/1506.01497.

[Scikit-Learn, 2019] Scikit-Learn (2019). Choosing the right estimator. https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html.

[StackOverflow, 2019] StackOverflow (2019). Stack overflow developer survey 2019. https://insights.stackoverflow.com/survey/2019.

[Sutton and Barto, 2018] Sutton, R. and Barto, A. (2018). Reinforcement Learning: An Introduction. Adaptive computation and machine learning. MIT Press.

[Tang et al., 1998] Tang, Y. Y., Tu, L.-T., Liu, J., Lee, S.-W., Lin, W.-W., and Shyu, I.-S. (1998). Offline recognition of chinese handwriting by multifeature and miltilevel classificaion. In IEEE Transactions on Pattern Analysis and Machine Intelligence (vol 20, no. 5), pages 556 – 561. IEEE.

[Tappert et al., 1990] Tappert, C. C., Suen, C. Y., , and Wakahara, T. (1990). State of the art in online handwriting. In IEEE Transactions On Pattern Analysis and Machine Intelligence (vol12 no.8), pages 787 – 808. IEEE.

[Verma and Ali, 2012] Verma, R. and Ali, D. J. (2012). A-survey of feature extraction and classification techniques in ocr systems. International Journal of Computer Applications & Information Technology, 1(3):1–3.

[WikiPython, 2019] WikiPython (2019). Integrating python with other languages. https://wiki.python.org/moin/IntegratingPythonWithOtherLanguages.

[Xu et al., 1992] Xu, L., Krzyzak, A., and Suen, C. Y. (1992). Methods of combining multiple classifiers and their applications to handwriting recognition. In IEEE Transactions on Systems, Man, and Cybernetics, pages 418 – 435. IEEE.

[Øivind Due Trier et al., 1996] Øivind Due Trier, Jain, A. K., and Taxt, T. (1996). Feature extraction methods for character recognition-a survey. Pattern Recognition, 29(4):641 – 662.